IBM Cúram Social Program Management
Versión 6.0.5

Cúram Express Rules - Manual de consulta

Nota

Antes de utilizar esta información y el producto al que hace referencia, lea la información que figura en el apartado "Avisos" en la página 265

Revisión: marzo de 2014

Esta edición se aplica a IBM Cúram Social Program Management v6.0.5 y a todos los releases posteriores mientras no se indique lo contrario en nuevas ediciones.

Contenido

Figuras

Tablas

Referencia de Cúram Express Rules

Las reglas de Cúram Express se utilizan para realizar cálculos de negocio. Se dispone de un entorno de desarrollo para la creación y prueba de los conjuntos de reglas de Cúram Express. Las reglas pueden ejecutarse en tiempo de ejecución. El editor CER es una herramienta para empresas y usuarios técnicos para ver, crear y gestionar conjuntos de reglas CER.

Introducción

Una descripción del idioma de la regla CER (Cúram Express Rules), el entorno de desarrollo y las características de ejecución.

Público al que va dirigido

Este manual de consulta debe ser consultado por cualquier persona implicada en el uso de CER para implementar las reglas de cálculo de empresa, incluyendo:

- Analistas empresariales, que están reuniendo los requisitos que implican cálculos empresariales; conocer las prestaciones y los enfoques de CER le ayudará a estructurar los requisitos de una forma que hace que la implementación en CER sea más sencilla;
- Desarrolladores de conjunto de reglas, que son responsables de codificar la lógica empresarial en un conjunto de reglas; tendrá que comprender el lenguaje de CER para codificar la lógica empresarial y
- Probadores, que son responsables de asegurar que la implementación cumple con los requisitos; necesitará conocer el soporte de pruebas de CER para decidir el enfoque de prueba.

Como puede darse el caso normalmente, cualquier persona determinada puede realizar más de uno de estos roles (si no todos). En función del rol y/o de los antecedentes, debe leer los capítulos de este manual en el orden que se adapte mejor a su caso.

Lectura relacionada

Tabla 1. Descripción de documentos relacionados

Documento	Tipo de documento	Cómo está relacionado
Working with Cúram Express Rules	Guía del desarrollador	Proporciona instrucciones paso a paso sobre cómo crear conjuntos de reglas CER en el editor CER utilizando ejemplos de varios grados de complejidad de reglas.
Inside Cúram Eligibility and Entitlement Using Cúram Express Rules	Guía del desarrollador	Describe cómo el motor de elegibilidad y titularidad de Cúram interactúa con CER para calcular las determinaciones para los casos de entrega de productos.

Tabla 1. Descripción de documentos relacionados (continuación)

Documento	Tipo de documento	Cómo está relacionado
How to Build a Product	Guía del desarrollador	Describe todas las tareas que deben realizarse como parte de la creación de un producto, incluyendo la asignación de reglas CER a un producto.
Universal Access Customization Guide	Guía del desarrollador	Describe cómo utilizar las reglas CER en el módulo de acceso universal de Cúram para proporcionar resultados de evaluación a los ciudadanos.

Estructura de este manual de consulta

Este manual es, en parte, una visión general y, en parte, material de consulta. *No* debe leerlo (necesariamente) del principio al final.

“Conformidad para CER” en la página 261
En este apéndice se explica cómo desarrollar con CER de una manera flexible.

Visión general de CER

Una breve visión general de CER, incluidos conceptos, ventajas y principios de orientación.

¿Qué es CER?

CER es:

- un lenguaje para expresar las reglas para los cálculos de negocio (en "conjuntos de reglas") y
- un entorno de desarrollo para crear y probar estos conjuntos de reglas.
- un entorno de ejecución para ejecutar reglas.

Lenguaje de reglas

CER es un lenguaje para definir las preguntas que se pueden plantear y las reglas para determinar las respuestas a estas preguntas.

Cada pregunta especifica:

- su nombre;
- el tipo de datos que proporciona la respuesta a la pregunta; y
- las reglas para proporcionar la respuesta (si se plantea la pregunta).

La respuesta a la pregunta puede ser tan simple como sí o no, por ejemplo, la pregunta "¿Esta persona es apta para recibir la prestación?". Sin embargo, puede definir tipos de respuesta que sean tan complejas como necesite, por ejemplo, la pregunta "¿Qué colectivos de personas de la unidad familiar tienen una necesidad urgente?" se puede responder proporcionando una lista de grupos de unidad familiar, con cada grupo de este tipo que contiene una lista de personas.

Las reglas para determinar la respuesta a la pregunta pueden ser de nuevo tan simples o complejas como necesite, por ejemplo, la regla para la respuesta a la pregunta "¿Cuál es la fecha de nacimiento del demandante?" es probable que (trivialmente) sea "la fecha que el demandante ha declarado que sea su fecha de nacimiento", mientras que la regla para responder a la pregunta "¿Es esta persona elegible para recibir una prestación?" probablemente implicará preguntas adicionales como "¿Qué nivel de ingresos tiene esta persona?" o "¿Cuántos hijos tiene esta persona?".

CER tiene su propia terminología para estos conceptos:

- **Clase de regla**

  Una clase de regla es un tipo de "cosa" que tiene datos como, por ejemplo, una Persona, unos Ingresos o una Reclamación. Se puede crear una clase de regla nueva en el Editor CER. Consulte “Vista técnica” en la página 105
- **Objeto de regla**

  Un objeto de regla es una instancia de una clase de regla, por ejemplo, Juan Pérez (una Persona), los ingresos de Juan Pérez de un trabajo a tiempo parcial (Ingresos) o la solicitud de Juan Pérez para recibir las prestaciones de pensión alimenticia (Reclamación).
- **Atributo de regla**

Un atributo de regla es una pregunta que se puede plantear. Se define en una clase de regla y se podría preguntar del cualquier objeto de regla de dicha clase, por ejemplo, la clase de regla Persona podría definir el atributo de regla dateOfBirth y al objeto de regla Juan Pérez se le podría preguntar su dateOfBirth, (por ejemplo, 3 de octubre de 1970). Se puede crear un nuevo atributo para la clase de regla seleccionada en el Editor CER. Consulte “Vista técnica” en la página 105

- **Expresión**

  Una expresión es un paso de cálculo que se puede utilizar para responder una pregunta, por ejemplo, si la idoneidad de una reclamación depende de que los ingresos totales de una persona estén por debajo de un determinado umbral, podemos utilizar la expresión "sum" para calcular los ingresos totales y después utilizar una expresión "compare" para comparar dicho total con el importe del umbral. Para crear una expresión, se puede arrastrar un elemento de regla "sum" al atributo de regla en el Editor CER. Consulte “Vista de empresa” en la página 104

- **Conjunto de reglas**

  Un conjunto de reglas es una colección de clases de regla, normalmente centradas alrededor de una finalidad específica, por ejemplo, un conjunto de reglas para una determinación de prestación por hijos podría incluir las clases de regla Reclamación, Persona e Ingresos. Un conjunto de reglas nuevo puede crearse en la sección de Reglas y pruebas de la interfaz de administración.

  Nota: Desde Cúram V6, los conjuntos de reglas ya no son autónomos. Una clase en un conjunto de reglas puede ampliar una clase de regla a partir de otro conjunto de reglas; el tipo de datos de un atributo de regla en un conjunto de reglas puede ser una clase de regla de otro conjunto de reglas; y las expresiones para leer o crear objetos de regla pueden utilizar clases de regla de otros conjuntos de reglas.

- **Sesión de reglas**

  Una sesión de regla controla la ejecución de las reglas. Por ejemplo, la aplicación podría crear una sesión de regla para determinar la idoneidad de Juan Pérez para la prestación por hijos, invocando al conjunto de reglas apropiado y preguntando preguntas de idoneidad sobre las circunstancias personales de Juan Pérez.

Entorno de creación y prueba

Los conjuntos de reglas CER se crean y se mantienen en el Editor CER. Los conjuntos de reglas CER se almacenan como datos XML en la base de datos de aplicación. Los datos XML para un conjunto de reglas CER se ajustan al esquema de reglas proporcionado por CER.

CER también incluye un validador de conjunto de reglas completo que puede detectar errores en el conjunto de reglas antes de permitir que se ejecuten las reglas. Puede validar el conjunto de reglas en el Editor CER. Consulte “Menú global de Editor CER” en la página 103

CER soporta las sesiones de regla en ejecución en:

- entornos de producción, donde CER se integra con la aplicación para responder a preguntas cuando es necesario y
- en un entorno de prueba autónomo donde se crean pruebas automatizadas repetibles para los conjuntos de reglas.

Los conjuntos de reglas CER son totalmente dinámicos. En entornos de producción, CER soporta la subida de los cambios en los conjuntos de reglas que entrarán en vigor cuando se publiquen; no se necesita la reconstrucción o el redespliegue de la aplicación.

La prueba de los conjuntos de reglas CER puede estar en cualquier nivel que se ajuste a sus necesidades. Puede elegir proporcionar datos de prueba detallados para un "escenario empresarial" completo y/o puede crear pruebas aisladas para partes del conjunto de reglas *sin* tener que configurar cuidadosamente grandes cantidades de datos de entrada.

Por ejemplo, la determinación de la elegibilidad de una persona para la prestación por hijos puede ser un cálculo complejo que implica (entre otras cosas) la comparación de los ingresos totales de la persona con un determinado umbral. Además, el cálculo de los ingresos totales de la persona es en *sí mismo* una cálculo complejo, que implica decisiones sobre si determinados tipos de ingresos son "contables" para la elegibilidad de la pensión alimenticia.

Al probar el cálculo de elegibilidad, en el desarrollo tradicional puede que tenga que configurar cuidadosamente los datos de ingresos para proporcionar un ingreso total calculado, lo que a su vez puede utilizar para probar el cálculo de elegibilidad. En función de la complejidad de los cálculos, esta configuración de datos puede ser una tarea muy tediosa y muy frágil de cambiar.

En comparación, en CER puede simplemente "arrancar" un cálculo sin tener que proporcionar datos detallados de bajo nivel; CER hace que sea extremadamente sencillo producir una prueba que efectivamente diga "para esta prueba, los ingresos totales son 10 $ - *no* intentar calcular los ingresos totales durante esta prueba".

Este recurso de CER hace que sea más fácil probar todas las funciones en el conjunto de reglas a un nivel que tenga sentido.

El entorno de creación también incluye herramientas para ayudar a desarrollar y probar las reglas:

- **RuleDoc**

  Extracto HTML de la estructura de los conjuntos de reglas.
- **SessionDoc**

  Representación HTML de los datos de los objetos de regla.
- **Herramienta de cobertura**

  Indica el alcance de su conjunto de reglas que han ejercido las pruebas.

Entorno de ejecución

CER ejecuta reglas a petición en el tiempo de ejecución.

Desde Cúram V6, CER también tiene características:

- para almacenar objetos de regla en la base de datos, de modo que los objetos de regla estén disponibles para el proceso futuro y
- para integrarse con el Gestor de dependencias a fin de detectar cuándo han cambiado los datos de entrada y actualizar automáticamente los resultados de cálculo que dependen de estos elementos de datos de entrada (semejante al proceso de hora cálculo familiar).

¿Cuáles son las ventajas?

CER ofrece estas ventajas clave:

- **Simplicidad**

  Los conjuntos de reglas CER son sólo tan complejos como los requisitos de la empresa. Los usuarios empresariales y los usuarios técnicos por igual pueden leer los conjuntos de reglas CER y entender fácilmente "lo que sucede". Los conjuntos de reglas son simples de escribir y sencillos de probar.

- **Flexibilidad**

  Los conjuntos de reglas son fáciles de cambiar. Puede añadir nuevas preguntas en cualquier momento y CER garantiza que el comportamiento existente no se vea afectado en absoluto. Puede realizar cambios en la manera en que se formula una pregunta existente y CER le mostrará qué cálculos dependen de esa pregunta, para que tenga todo el control del efecto del cambio.

- **Soporte de localización**

  CER puede producir salida localizable, para que las respuestas a las preguntas se pueden visualizar a los usuarios finales según sus preferencias de idioma y entorno local.

- **Validez**

  CER trabaja arduamente para detectar errores en el conjunto de reglas antes de ejecutarlo. El validador de conjunto de reglas CER informa de tantos errores como puede para que éstos se puedan arreglar de una vez. CER busca problemas técnicos en el conjunto de reglas para que sólo se tenga que concentrar en la funcionalidad del conjunto de reglas.

- **Contrastabilidad**

  Puede probar los conjuntos de reglas CER en la granularidad que se adapte mejor a sus necesidades. CER le permite mantener el control sobre los conjuntos de reglas grandes creando pruebas para secciones separadas de las reglas.

- **Soporte dinámico**

  Puede realizar cambios en el conjunto de reglas CER en un sistema en ejecución y los cambios entrarán en vigor inmediatamente una vez realizada la publicación.

- **Comportamiento parecido al de una hoja de cálculo**

  La construcción de reglas CER es similar a la creación de capas de fórmulas en las casillas de una hoja de cálculo (que será familiar para muchos usuarios). Cuando los datos de entrada (por ejemplo pruebas, datos personales o tarifas de pago) cambian, CER se integra con el Gestor de dependencias para volver a calcular automáticamente los valores derivados que se ven afectados por el cambio.

¿Cómo se puede utilizar?

El entorno de desarrollo de CER se integra firmemente con el entorno de desarrollo de aplicaciones más amplio.

CER se utiliza en varias áreas de aplicación, incluyendo (pero sin limitarse a):

- **Módulo de acceso universal de Cúram**

  El módulo de acceso universal de Cúram se basa en CER para determinar la elegibilidad potencial para programas de servicios sociales cuando los ciudadanos utilizan el módulo de autoservicio para realizar la autoevaluación. Basándose en los datos capturados, CER determina para qué programas es

potencialmente elegible el ciudadano y proporciona una explicación textual (en el idioma del ciudadano) de *por qué* ese ciudadano es o no es potencialmente elegible.

- **Asesor**

  El Asesor utiliza reglas CER para calcular los consejos a mostrar a los usuarios. Estos consejos se actualizan automáticamente cuando cambian las circunstancias.

- **Motor de elegibilidad y titularidad**

  El Motor de elegibilidad y titularidad de Cúram se integra estrechamente con CER para proporcionar determinaciones de la elegibilidad y titularidad de un caso (y explicaciones de cómo éstas se han obtenido) a lo largo del tiempo de vida completo del caso. El Motor de elegibilidad y titularidad de casos se basa en la integración entre CER y el Gestor de dependencias para identificar cuándo se debe volver a calcular la determinación de un caso, ya sea debido a cambios específicos de caso como datos de pruebas, ya sea debido a los cambios de datos más amplios, como cambios en los datos personales o datos de tipo de todo el producto.

También puede utilizar CER para realizar cálculos para sus propias áreas empresariales personalizadas.

El tiempo de ejecución de CER no tiene ningún concepto empresariales incorporado; en su lugar, cada área empresarial se comunica con CER mediante el uso de:

- clases de regla de interfaz que encapsulan los conceptos empresariales y/o
- extensiones del lenguaje de CER que son conscientes del concepto de empresa.

Para obtener detalles sobre cómo los componentes de aplicación utilizan CER, consulte las guías empresarial y técnicas de dicho componente.

Principios de guía

En su núcleo, CER mantiene ciertos principios clave. Conocer un poco estos principios puede ayudarle a entender el enfoque de CER:

- **Las preguntas sólo se responden cuando se necesitan**

  El trabajo realizado para responder a una pregunta sólo se realiza cuando se plantea esa pregunta.

- **Todos los datos son inmutables**

  La respuesta a una pregunta es un valor que no puede cambiarse accidentalmente por algo externo a CER. Si se vuelve a calcular la respuesta a una pregunta, se creará un valor de respuesta nuevo.

- **Permitir varias preguntas**

  Un conjunto de reglas no se ejecuta una vez por completo; en lugar de ello, un conjunto de reglas permite plantear tantas preguntas como es necesario.

- **Especificar reglas, no orden de ejecución**

  Especifique las reglas para responder a una pregunta y dejar que CER responda de manera eficiente a esas preguntas en el tiempo de ejecución.

- **Sin volatilidad**

  Entradas idénticas procesadas por reglas idénticas producen siempre la misma salida.

- **Sin "almacenamiento de trabajo"**

No hay contadores o totales acumulados. Un recuento o total es una pregunta por derecho propio: se proporcionan las reglas para responderla y CER se ocupa del orden de ejecución cuando responde.

- **Nombrar lo que se necesita**

  Sólo tiene que proporcionar nombres para conceptos empresariales y preguntas. No tiene que imaginar nombres descriptivos para resultados provisionales (a menos que lo desee).

- **Desarrollo y prueba van de la mano**

  CER proporciona un sólido soporte para gestionar las pruebas de las reglas.

- **Sin conceptos empresariales incorporados**

  El tiempo de ejecución de CER no contiene adrede conceptos empresariales. Defina los conceptos empresariales que necesita, dejando que el tiempo de ejecución de CER sea un entorno de uso general.

- **La implementación de reglas se correlaciona cuidadosamente con los requisitos de regla**

  La implementación de los requisitos de regla es tan compleja como esos requisitos: *pero no más*. Los conjuntos de reglas CER "tienen sentido" cuando los ven los analistas empresariales que han reunido los requisitos originales.

- **Leer acerca del soporte de Java conocido públicamente**

  CER no reinventa la rueda: la funcionalidad proporcionada por la tecnología Java™ existente se reutiliza fácilmente en los conjuntos de reglas CER.

- **Gestión de dependencias de cálculo**

  CER se integra con el Gestor de dependencias para gestionar dependencias de cálculo para que el usuario no tenga que hacerlo. Cuando cambia el elemento de datos de entrada, el Gestor de dependencias y CER saben qué se debe recalcular. No tiene que escribir ningún proceso especial que calcula qué salidas calculadas *pueden* estar afectadas.

Herramientas de desarrollo y prueba de CER

Las herramientas que están disponibles para desarrollar y probar los conjuntos de reglas CER.

Editor de reglas CER

El editor CER proporciona un entorno y una interfaz agradables para el usuario para que ambos usuarios, técnicos y empresariales, creen, editen y validen un conjunto de reglas y sus clases de regla.

Si desea más información sobre cómo utilizar el Editor CER, consulte “Acerca del Editor CER” en la página 102.

Soporte de localización

Una descripción de la localización en CER.

La localización en CER cubre estas dos tareas diferentes:

- localización de datos calculados devueltos por un atributo de regla CER, de modo que la salida se pueda visualizar a los usuarios en distintos entornos locales y
- localización de las descripciones para artefactos de los conjuntos de reglas CER, para que los usuarios que ven los conjuntos de reglas en el Editor CER puedan hacerlo en su propio entorno local.

Estos elementos se describen más detalladamente en las secciones siguientes.

Importante: Los *nombres* de elementos de conjunto de reglas, por ejemplo como clases de regla y atributos, *no se pueden* convertir a entornos locales, porque se utilizan en el lenguaje CER como identificadores, por ejemplo el nombre de un atributo en una expresión `reference`.

En cambio, las *descripciones* de los elementos de regla se pueden convertir al entorno local, dejando los nombres de elementos sin cambiar.

Localización de datos calculados

CER soporta la clase Java habitual `String`.

Las series pueden ser útiles en las fases iniciales del desarrollo del conjunto de reglas; sin embargo, si las reglas contienen salida que debe visualizarse a los usuarios en configuraciones regionales diferentes, es posible que tenga que aprovechar el soporte de localización de CER.

El valor de la serie "Hello World" en este ejemplo es correcto para los usuarios que leen inglés, pero ¿qué sucede si no?

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="HelloWorldRuleSet"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="HelloWorld">

    <Attribute name="greeting">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <String value="Hello, world!"/>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

CER incluye una interfaz `curam.creole.value.Message` que permite la conversión de un valor en una serie específica de configuración regional en tiempo de ejecución.

Para obtener una lista de expresiones de CER que pueden crear una instancia de `curam.creole.value.Message`, consulte "Mensajes traducibles" en la página 163.

Ahora vamos a reescribir el ejemplo `HelloWorld` de modo que sea traducible:

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="LocalizableHelloWorldRuleSet"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="HelloWorld">

    <Attribute name="greeting">
      <type>
        <!-- Utilizar mensaje, no serie -->
        <javaclass name="curam.creole.value.Message"/>
      </type>
      <derivation>
```

```
        <!-- Buscar el valor de una propiedad
             traducible, en lugar de grabar en el código fuente
             una serie de un solo idioma -->
        <ResourceMessage key="greeting"
          resourceBundle="curam.creole.example.HelloWorld"/>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

Localización de “Hello, world!” en inglés.

```
# file curam/creole/example/HelloWorld_en.properties

greeting=Hello, world!
```

Localización de “Hello, world!” en francés.

```
# file curam/creole/example/HelloWorld_fr.properties

greeting=Bonjour, monde!
```

¿Cómo se comportará este mensaje en el tiempo de ejecución? Cualquier código que interactúa con el conjunto de reglas debe invocar el método `toLocale` en todos los mensajes, para convertirlos al entorno local necesario.

Este ejemplo muestra la interacción con el conjunto de reglas localizado.

```
package curam.creole.example;

import java.util.Locale;

import junit.framework.TestCase;
import curam.creole.execution.session.RecalculationsProhibited;
import curam.creole.execution.session.Session;
import curam.creole.execution.session.Session_Factory;
import
 curam.creole.execution.session.StronglyTypedRuleObjectFactory;
import
 curam.creole.ruleclass.LocalizableHelloWorldRuleSet.impl.HelloWorld;
import
 curam.creole.ruleclass.LocalizableHelloWorldRuleSet.impl.HelloWorld_Factory;
import curam.creole.storage.inmemory.InMemoryDataStorage;
import curam.creole.value.Message;

public class TestLocalizableHelloWorld extends TestCase {

  /**
   * Ejecuta la clase como aplicación Java autónoma.
   */
  public static void main(final String[] args) {

    final TestLocalizableHelloWorld testLocalizableHelloWorld =
        new TestLocalizableHelloWorld();
    testLocalizableHelloWorld.testLocalizedRuleOutput();

  }

  /**
   * Un caso de prueba simple, que visualiza salida localizada en diferentes
   * entornos locales.
   */
  public void testLocalizedRuleOutput() {

    final Session session =
```

```
        Session_Factory.getFactory().newInstance(
            new RecalculationsProhibited(),
            new InMemoryDataStorage(
                new StronglyTypedRuleObjectFactory()));

    final HelloWorld helloWorld =
        HelloWorld_Factory.getFactory().newInstance(session);

    // devuelve un mensaje, no una serie
    final Message greeting = helloWorld.greeting().getValue();

    // para decodificar el mensaje, tenemos que utilizar el entorno local del usuario
    final String greetingEnglish =
        greeting.toLocale(Locale.ENGLISH);
    final String greetingFrench = greeting.toLocale(Locale.FRENCH);

    System.out.println(greetingEnglish);
    System.out.println(greetingFrench);

    assertEquals("Hello, world!", greetingEnglish);
    assertEquals("Bonjour, monde!", greetingFrench);
  }

}
```

Si se utiliza en un mensaje traducible, los tipos de datos siguientes se formatean en el tiempo de ejecución de una forma que tiene en cuenta el entorno local:

- Objetos de regla (utilizando el valor del atributo `description` del objeto de regla
- Fechas (utilizando `curam.util.type.Date`);
- Elementos de tabla de código y
- mensajes localizables anidados.

Todos los demás objetos se visualizan utilizando su método `toString`.

Localización de descripciones de artefacto de regla CER

El Editor CER le permite colocar una descripción de estos artefactos de conjunto de reglas (a través de la anotación "Etiqueta"):

- Conjunto de reglas;
- Clase de regla
- Atributo de regla y
- Expresión.

Cuando un conjunto de reglas CER se publica utilizando la aplicación de administración de Cúram, las descripciones de estos artefactos de conjunto de reglas se almacenan en el almacén de recursos de la aplicación como archivos de propiedades (denominado `RULESET- (nombre de conjunto de reglas) (número de versión del conjunto de reglas)`. Puede localizar estos archivos de propiedades igual que cualquier otro recurso del almacén de recursos.

El soporte para localizar artefactos de reglas CER mediante el Editor CER se incluirá en un release futuro del Editor CER.

Validador de conjunto de reglas

CER contiene un validador que comprueba la estructura de los conjuntos de reglas. Normalmente, validará los conjuntos de reglas utilizando la aplicación de administración de Cúram o el editor de CER.

Para los conjuntos de reglas en el sistema de archivos, puede ejecutar este mandato para validar la estructura de estos conjuntos de reglas:

```
build creole.validate.rulesets
```

El destino ejecutará el validador de conjunto de reglas CER en los conjuntos de reglas para informar de los errores y/o avisos acerca de los conjuntos de reglas CER.

Consejo: El validador de conjunto de reglas CER también informará de los avisos sobre problemas no muy graves en los conjuntos de reglas. Estos avisos no impiden que se ejecuten y prueben las reglas, pero deben tratarse de manera que se garantice un conjunto de reglas óptimo.

Intérprete de conjunto de reglas

CER contiene un intérprete que puede ejecutar conjuntos de reglas definidos dinámicamente.

Este código de ejemplo utiliza el intérprete de conjunto de reglas CER para ejecutar reglas desde `HelloWorldRuleSet`.

```
package curam.creole.example;

import junit.framework.TestCase;
import curam.creole.execution.RuleObject;
import curam.creole.execution.session.InterpretedRuleObjectFactory;
import curam.creole.execution.session.RecalculationsProhibited;
import curam.creole.execution.session.Session;
import curam.creole.execution.session.Session_Factory;
import curam.creole.parser.RuleSetXmlReader;
import curam.creole.ruleitem.RuleSet;
import curam.creole.storage.inmemory.InMemoryDataStorage;

public class TestHelloWorldInterpreted extends TestCase {

  /**
   * Ejecuta la clase como aplicación Java autónoma.
   */
  public static void main(final String[] args) {

    final TestHelloWorldInterpreted testHelloWorld =
        new TestHelloWorldInterpreted();
    testHelloWorld.testUsingInterpreter();
  }

  /**
   * Lee HelloWorldRuleSet del archivo de origen XML.
   */
  private RuleSet getRuleSet() {

    /* La vía de acceso relativa al archivo de origen de conjunto de reglas */
    final String ruleSetRelativePath = "./rules/HelloWorld.xml";

    /* leer en el origen de conjunto de reglas */
    final RuleSetXmlReader ruleSetXmlReader =
        new RuleSetXmlReader(ruleSetRelativePath);

    /* volcar los problemas */
    ruleSetXmlReader.validationProblemCollection().printProblems(
        System.err);

    /* fallar si hay errores en el conjunto de reglas */
    assertTrue(!ruleSetXmlReader.validationProblemCollection()
```

```
        .containsErrors());

    /* devolver el conjunto de reglas del lector */
    return ruleSetXmlReader.ruleSet();
  }

  /**
   * Un caso de prueba simple, utilizando el intérprete de conjunto de reglas CER
   * totalmente dinámico.
   */
  public void testUsingInterpreter() {

    /* leer en el conjunto de reglas */
    final RuleSet ruleSet = getRuleSet();

    /* iniciar una sesión que crea objetos de regla interpretados */
    final Session session =
        Session_Factory.getFactory().newInstance(
            new RecalculationsProhibited(),
            new InMemoryDataStorage(
                new InterpretedRuleObjectFactory()));

    /* crear una instancia de objeto de regla de la clase de regla necesaria */
    final RuleObject helloWorld =
        session.createRuleObject(ruleSet.findClass("HelloWorld"));

    /*
     * Acceder al atributo de regla de saludo ("greeting") en el objeto de regla
     * el resultado se debe convertir al tipo esperado (String)
     */
    final String greeting =
        (String) helloWorld.getAttributeValue("greeting")
            .getValue();

    System.out.println(greeting);
    assertEquals("Hello, world!", greeting);
  }

}
```

Ahora puede ejecutar esta clase de ejemplo como una aplicación Java autónoma (es decir, a través de su método `principal`) o como una prueba JUnit. Estas dos maneras de ejecutar esta clase se proporcionan sólo por comodidad, de modo que cuando escriba su propio código para ejecutar los conjuntos de reglas, elija el que mejor se adapte a sus necesidades.

Ahora vamos a profundizar en los detalles de este código. El método de prueba `testUsingInterpreter` realiza estas funciones clave:

- lee en el conjunto de reglas;
- inicia una sesión CER;
- crea una nueva instancia de objeto de regla en la sesión y
- ejecuta reglas recuperando el valor de un atributo del objeto de regla.

Leer en el conjunto de reglas

```
/* leer en el conjunto de reglas */
             final RuleItem_RuleSet ruleSet = getRuleSet();
```

Esta línea llama a un método de programa de utilidad para leer el conjunto de reglas desde un archivo de origen XML.

El conjunto de reglas se valida explícitamente para asegurarse de que no contiene errores:

```
/* volcar los problemas */
             ruleSetXmlReader.validationProblemCollection().printProblems(
             System.err);

             /* fallar si hay errores en el conjunto de reglas */
             assertTrue(!ruleSetXmlReader.validationProblemCollection()
             .containsErrors());
```

Iniciar una sesión CER

```
/* iniciar una sesión que crea objetos de regla interpretados */
             final Session session =
             Session_Factory.getFactory().newInstance(
             new RecalculationsProhibited(),
             new InMemoryDataStorage(
             new InterpretedRuleObjectFactory()));
```

Estas líneas crean una sesión CER nueva para el conjunto de reglas.

Una sesión gestiona los objetos de regla creados para las clases del conjunto de reglas. En este ejemplo, estamos utilizando una sesión que crea objetos de regla totalmente dinámicos (utilizando `InterpretedRuleObjectFactory`); como veremos a continuación, en una sesión interpretada, cada referencia a una clase de regla o nombre de atributo es a través de una llamada de API que toma ese nombre como un parámetro String.

Crear un nuevo objeto de regla

```
/* crear una instancia de objeto de regla de la clase de regla necesaria */
             final RuleObject helloWorld =
             session.createRuleObject("HelloWorld");
```

Esta línea crea un objeto de regla nuevo (una instancia de la clase de regla "HelloWorld") y almacena el objeto de regla en la memoria de la sesión de CER.

Ejecutar reglas

```
/*
   * Acceder al atributo de regla de saludo ("greeting") en el objeto de regla
   * el resultado se debe convertir al tipo esperado (String)
   */
   final String greeting =
   (String) helloWorld.getAttributeValue("greeting")
   .getValue();
```

Esta línea recupera el valor del atributo de saludo ("greeting") del objeto de regla creado anteriormente.

Cuando se solicita el valor del atributo, CER ejecuta las reglas para obtener el valor del atributo (en este caso, devolviendo la serie constante "Hello, world!").

Nota: Cuando se ejecuta una sesión interpretada, debe convertir el resultado de `getValue` al tipo de datos esperado.

En este ejemplo, sólo hemos solicitado el valor de un atributo; sin embargo, mientras la sesión aún está activa, el código puede solicitar el valor de cualquier atributo de cualquier objeto de regla de la sesión. CER recuerda los valores ya calculados y sólo realiza un cálculo de la primera vez que se solicita.

Generador de códigos de prueba de CER

CER se incluye con un generador de códigos que puede generar clases de derivador Java para las clases de regla. Estas clases generadas pueden simplificar

la escritura del código de prueba y permitir que el compilador detecte problemas que, de lo contrario, sólo se producirían en el tiempo de ejecución.

El intérprete del conjunto de reglas CER permite hacer referencia a la clase de regla y a los nombres de atributo a través de series. Mientras que esto permite la configuración totalmente dinámica de los conjuntos de reglas, para realizar pruebas puede ser engorroso tener que utilizar series y los valores de atributo de conversión. Si especifica una clase de regla o un nombre de atributo de forma incorrecta, o utiliza el tipo incorrecto de conversión de tipo de datos, el código puede compilarse sin errores, pero obtendrá errores durante la ejecución.

Ahora para volver a escribir nuestro código para la ejecución de HelloWorldRuleSet para mostrar las reglas de ejecución con las clases de regla de prueba generadas por CER.

```
package curam.creole.example;

import junit.framework.TestCase;
import curam.creole.execution.session.RecalculationsProhibited;
import curam.creole.execution.session.Session;
import curam.creole.execution.session.Session_Factory;
import
 curam.creole.execution.session.StronglyTypedRuleObjectFactory;
import curam.creole.ruleclass.HelloWorldRuleSet.impl.HelloWorld;
import
 curam.creole.ruleclass.HelloWorldRuleSet.impl.HelloWorld_Factory;
import curam.creole.storage.inmemory.InMemoryDataStorage;

public class TestHelloWorldCodeGen extends TestCase {

  /**
   * Ejecuta la clase como aplicación Java autónoma.
   */
  public static void main(final String[] args) {

    final TestHelloWorldCodeGen testHelloWorld =
        new TestHelloWorldCodeGen();
    testHelloWorld.testUsingGeneratedTestClasses();

  }

  /**
   * Un caso de prueba simple, utilizando las clases de prueba generadas por CER para
   * tipificación firme y facilidad de codificación de pruebas.
   */
  public void testUsingGeneratedTestClasses() {

    /* iniciar una sesión de tipificación firme */
    final Session session =
        Session_Factory.getFactory().newInstance(
            new RecalculationsProhibited(),
            new InMemoryDataStorage(
                new StronglyTypedRuleObjectFactory()));

    /*
     * crear una instancia de objeto de regla de la clase de regla necesaria,
     * utilizando la fábrica generada
     */
    final HelloWorld helloWorld =
        HelloWorld_Factory.getFactory().newInstance(session);

    /*
     * utilizar el descriptor de acceso generado para llegar al atributo de regla de saludo ("gree
     * - ninguna conversión es necesaria y cualquier error en el
     * nombre de atributo llevará a un error de compilación
```

```
     */
    final String greeting = helloWorld.greeting().getValue();

    System.out.println(greeting);
    assertEquals("Hello, world!", greeting);
  }

}
```

Observe las comparaciones siguientes con nuestro código `TestHelloWorldInterpreted`:

- La sesión utilizada con el generador de códigos utiliza `StronglyTypedRuleObjectFactory`, que se denomina así porque trata con clases Java generadas en lugar de instancias `RuleObject` dinámicas;
- la carga del conjunto de reglas no es necesario (por lo tanto ningún método de programa de utilidad);
- la referencia a la clase de regla `HelloWorld` es a través de una interfaz Java con un nombre idéntico; el compilador Java detectará cualquier error tipográfico en el nombre;
- de forma similar, la referencia al atributo de regla `greeting` es a través de un método Java con un nombre idéntico en la interfaz;
- no se necesita ninguna conversión del tipo de retorno, porque el método `greeting` generado devuelve el tipo correcto (`String`).

aviso: El código generado sólo está pensado para utilizarse en entornos de prueba donde es un asunto sencillo para volver a compilar los cambios en el código.

El código generado *no* es portable entre máquinas, porque contiene vías de acceso absolutas a los conjuntos de reglas en la máquina local.

En concreto, *no* debe utilizar el código generado en cualquier entorno de producción donde los conjuntos de reglas pueden cambiar dinámicamente.

Cómo generar código

Para ejecutar el generador de códigos CER, ejecute el mandato siguiente:

```
build creole.generate.test.classes
```

El destino también ejecutará el validador de conjunto de reglas CER en los conjuntos de reglas. Si hay errores, el validador de conjunto de reglas CER informa de los errores y el proceso se detiene. Si no hay errores, el generador de CER producirá interfaces y clases Java generadas para los conjuntos de reglas y clases de regla CER.

Consejo: El validador de conjunto de reglas CER también informará de los avisos sobre problemas no muy graves en los conjuntos de reglas. Estos avisos no impiden que se ejecuten y prueben las reglas, pero deben tratarse de manera que se garantice un conjunto de reglas óptimo.

El generador de código de CER colocará la salida en el directorio `EJBServer/build/svr/creole.gen/source`.

Esto es un ejemplo de interfaz Java generada para la clase de regla `HelloWorld`:

```
/*
 * Generado por el generador de código CREOLE de Curam
 * Generator Copyright 2008-2010 Curam Software Ltd.
```

```
 */
package curam.creole.ruleclass.HelloWorldRuleSet.impl;
/**
 * Interfaz generada por código para pruebas.
 * <p/>
 * Los clientes no deben implementar esta interfaz.
 */
public interface HelloWorld extends
 curam.creole.execution.RuleObject {
  /**
   * Descriptor de acceso generado por código para pruebas.
   * @devolver contenedor para el valor de atributo de saludo
   */
  public curam.creole.execution.AttributeValue<? extends
 java.lang.String> greeting();
}
```

Consejo: Debe volver a generar las clases de prueba si se realizan cambios estructurales en los conjuntos de reglas en el sistema de archivos, por ejemplo

- crear un conjunto de reglas nuevo o eliminar un conjunto de reglas existente;
- añadir una nueva clase de regla a un conjunto de reglas o eliminar una clase de regla existente a un conjunto de reglas;
- añadir un atributo de regla nuevo a una clase de regla o eliminar un atributo de regla existente de una clase de regla;
- cambiar el valor "extends" para una clase de regla existente y/o
- cambiar el tipo de datos de un atributo.

No necesita volver a generar las clases de prueba si los cambios están limitados a la *implementación* de un atributo de regla (es decir las expresiones de derivación). Las derivaciones se procesan siempre de forma dinámica desde el conjunto de reglas durante el tiempo de ejecución y no están presentes en las clases de prueba generadas.

Herramienta de cobertura de conjunto de reglas

CER incluye una herramienta para informar sobre las partes de un conjunto de reglas que están "cubiertas" en el tiempo de ejecución.

Se puede informar de las estadísticas de cobertura para cualquier proceso que solicite valores de CER, incluyendo:

- la aplicación en línea en ejecución y
- ejecuciones de pruebas JUnit.

Para capturar los datos de cobertura, establezca la propiedad de entorno `curam.creole.coverage.logfile` (en `Bootstrap.properties`) en la ubicación de un archivo. Mientras se ejecutan las reglas, las líneas que contienen información de cobertura se añaden al archivo cuando se evalúan las expresiones de CER.

Consejo: Para borrar los datos de cobertura, simplemente suprima el archivo especificado en el valor `curam.creole.coverage.logfile`.

Con el tiempo, el archivo de datos de cobertura puede aumentar mucho de tamaño, por lo tanto deberá desactivar la captura de datos de cobertura cuando no sea necesaria, eliminando (o comentando) el valor `curam.creole.coverage.logfile`.

Para crear un informe de cobertura, ejecute el siguiente destino:

build creole.report.coverage -Dfile.coverage.log= *ubicación de archivo*

Se grabará un informe simple sondeable codificado en colores en `.../EJBServer/build/svr/creole.gen/coverage/index.html`, donde

- Verde = cubierto
- Amarillo = parcialmente cubierto
- Rojo = no cubierto

Los atributos de regla con una derivación de <especificado> se excluyen adrede del informe. A continuación se muestra un informe de ejemplo:

CREOLE Coverage Report

Generated: 23-Mar-2011 16:33:07

Coverage for Rule Set: SimpleTestProductEligibilityEntitlementRuleSet

Rule Class Name	Rule Attribute Name	Rule Expressions	Fully Covered		Partially Covered		Not Covered	
Rule Set Summary		623	231	37.08%	1	0.16%	391	62.76%
AgeRangeCalculator		*47*	*45*	*95.74%*	*0*	*0.00%*	*2*	*4.26%*
	description	2	0	0.00%	0	0.00%	2	100.00%
	homeHelpAgeTimeline	5	5	100.00%	0	0.00%	0	0.00%
	isAliveTimeline	10	10	100.00%	0	0.00%	0	0.00%
	isHomeHelpAgeTimeline	8	8	100.00%	0	0.00%	0	0.00%
	isInAgeRangeTimeline	10	10	100.00%	0	0.00%	0	0.00%
	maximumAge	1	1	100.00%	0	0.00%	0	0.00%
	maximumAgeTimeline	5	5	100.00%	0	0.00%	0	0.00%
	minimumAge	1	1	100.00%	0	0.00%	0	0.00%
	minimumAgeTimeline	5	5	100.00%	0	0.00%	0	0.00%
	personCalculator	0	0		0		0	
CREOLEBonus		*0*	*0*		*0*		*0*	
	amount	0	0		0		0	
	evidenceID	0	0		0		0	
	type	0	0		0		0	
CREOLEBonusCalculator		*5*	*3*	*60.00%*	*0*	*0.00%*	*2*	*40.00%*

Figura 1. Ejemplo de informe de cobertura

Tenga en cuenta que los conjuntos de reglas y las clases que están "incluidos" en otros conjuntos de reglas (utilizando el mecanismo <Include>) pasan a formar parte esencialmente del origen del conjunto de reglas de inclusión más externo. Se deberá tener en cuenta esto al analizar los informes de cobertura.

Área de publicación de CER

La Aplicación de administración de Cúram contiene pantallas para listar los conjuntos de reglas CER.

Desde aquí, puede:

- ver un conjunto de reglas existente (en el editor CER) y ver versiones históricas de cualquier conjunto de reglas CER;
- una vez abierto, realizar cambios en un conjunto de reglas existente (en el editor CER);

- crear un conjunto de reglas nuevo (y abrirlo en el editor CER para añadir reglas) y/o
- eliminar un conjunto de reglas existente.

Desde Cúram V6, los cambios en los conjuntos de reglas CER *no* entran en vigor inmediatamente, sino que en lugar de ello se almacenan en la área de publicación hasta que se publican.

Puede acumular muchos cambios en los conjuntos de reglas CER en esta área; en realidad, puede *tener* que acumular cambios en muchos conjuntos de reglas, si el cambio que está realizando afecta a más de un conjunto de reglas.

Puede elegir validar los cambios pendientes en cualquier momento. Cuando esté satisfecho con los cambios, puede elegir publicarlos. El sistema revalidará que los conjuntos de reglas son válidos y, si es así, permitirá la publicación para continuar.

La publicación de los cambios de conjunto de reglas CER se produce en proceso aplazado, porque los objetos de regla CER existentes se tendrán que actualizar según los cambios en las clases de reglas CER y/o los recálculos en cola para atributos cuyas derivaciones han cambiado.

Generación de esquemas y catálogos

El esquema para los conjuntos de reglas CER se ensambla dinámicamente para tener en cuenta el esquema fijo para los conjuntos de reglas, las clases y los atributos de CER; y las contribuciones a las expresiones y anotaciones de CER mediante componentes de aplicación.

Normalmente el esquema ensamblado dinámicamente reside en la memoria cuando CER realiza el proceso de validación. Sin embargo, en algunas ocasiones, puede ser útil tener este esquema (y un catálogo apuntando a él) en el sistema de archivos.

Para generar el archivo de esquema CER (`EJBServer/build/svr/creole.gen/schema/RuleSet.xsd`), ejecute el mandato siguiente:

```
build creole.generate.schema
```

Para generar un catálogo (`EJBServer/build/svr/creole.gen/catalog/CREOLECatalog.xml`) que apunte al archivo de esquema CER, ejecute el mandato siguiente:

```
build creole.generate.catalog
```

RuleDoc

RuleDoc es un conjunto de documentación que puede generar automáticamente desde los conjuntos de reglas y las clases de regla de Cúram Express Rules (CER). CER proporciona una herramienta para generar RuleDoc.

RuleDoc es útil para las tareas siguientes:

- Discutir el comportamiento del conjunto de reglas CER con un público no técnico.
- Visualizar las dependencias entre los atributos de regla, particularmente, a medida que los conjuntos de reglas crecen en complejidad.

- Comprender los impactos de los cambios que realice en una derivación de un atributo de regla.

Ejemplo simple

A continuación se proporciona un XML para el conjunto de reglas simple Hello Worldrule:

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="HelloWorldRuleSet"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="HelloWorld">

    <Attribute name="greeting">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <String value="Hello, world!"/>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

A continuación se muestra el RuleDoc generado para el conjunto de reglas anterior, listando la clase de regla individual:

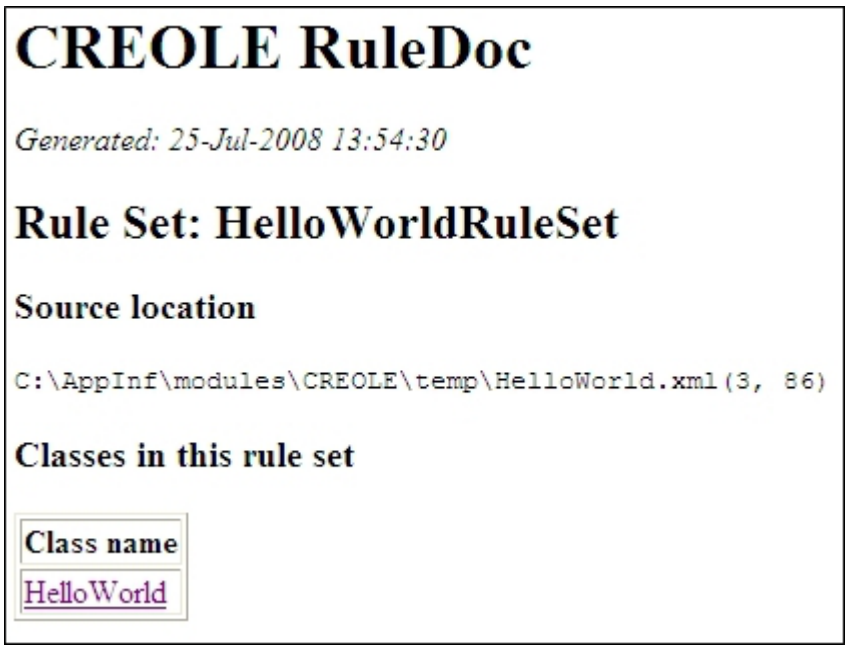

Figura 2. RuleDoc para `HelloWorldRuleSet`

Al pulsar en la clase de regla `HelloWorld` se muestra el RuleDoc:

Figura 3. RuleDoc para la clase de regla `HelloWorld`

Y al pulsar en el atributo `greeting`, se desplaza a su derivación:

Figura 4. RuleDoc para el atributo de regla `greeting`

Un ejemplo más útil

Para conjuntos de reglas más complejos, RuleDoc puede ayudarle a:

- navegar por las dependencias entre las clases de regla del conjunto de reglas;
- entender el cálculo de derivación de cada atributo de regla y
- ver qué otros atributos de regla dependen de un atributo de regla concreto.

A continuación se proporciona el XML para un conjunto de reglas más complejas para una calculadora de un año de jubilación:

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="RetirementYearRuleSet"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
```

```
  <Class name="RetirementYear">

    <Attribute name="yearOfBirth">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <Number value="1970"/>
      </derivation>
    </Attribute>

    <Attribute name="ageAtRetirement">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <Number value="65"/>
      </derivation>
    </Attribute>

    <Attribute name="yearOfRetirement">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <arithmetic operation="+">
          <reference attribute="yearOfBirth"/>
          <reference attribute="ageAtRetirement"/>
        </arithmetic>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

A continuación se muestra el RuleDoc generado para el conjunto de reglas de la calculadora de un año de jubilación:

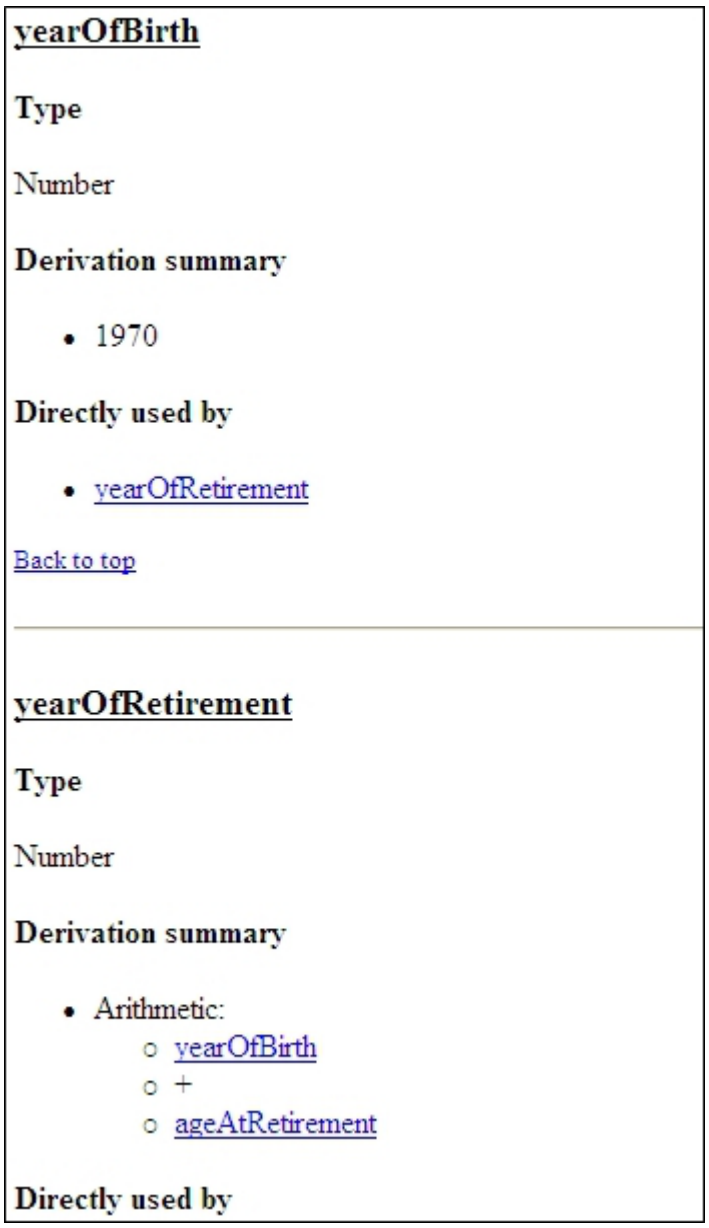

Figura 5. RuleDoc que muestra derivación y uso

Este ejemplo muestra:

- la derivación del atributo de regla `yearOfRetirement` (con enlaces a los atributos de regla `yearOfBirth` y `ageAtRetirement` de los que depende) y
- el atributo de regla `yearOfBirth`, con un enlace al atributo de regla `yearOfRetirement` que depende de él directamente.

Cómo generar RuleDoc

Para ejecutar el generador RuleDoc de CER, ejecute el mandato siguiente:

```
build creole.generate.ruledoc
```

El destino también ejecutará el validador de conjunto de reglas CER en los conjuntos de reglas. Si hay errores, el validador de conjunto de reglas CER informa

de los errores y el proceso se detiene. Si no hay errores, el generador de CER producirá RuleDoc para los conjuntos de reglas y las clases de regla.

Consejo: El validador de conjunto de reglas CER también informará de los avisos sobre problemas no muy graves en los conjuntos de reglas. Estos avisos no impiden que se ejecuten y prueben las reglas, pero deben tratarse de manera que se garantice un conjunto de reglas óptimo.

El generador RuleDoc de CER pondrá la salida en el directorio `EJBServer/build/svr/creole.gen/ruledoc`.

SessionDoc

Puede generar una documentación HTML de salida llamada SessionDoc durante una sesión. SessionDoc proporciona un registro de todos los objetos de regla creados durante la sección y puede ser una sólida ayuda de depuración cuando se utiliza junto con sus pruebas.

Puede ser útil utilizar el punto de enlace `tearDown` para forzar que la salida de todos los métodos de prueba de la clase de prueba esté en su SessionDoc:

```
@Override
          protected void tearDown() throws Exception {
          /*
          * Escribir SessionDoc, en un directorio denominado como el
          * método de prueba.
          */
          final File sessionDocOutputDirectory =
          new File("./gen/sessiondoc/" + this.getName());
          sessionDoc.write(sessionDocOutputDirectory);

          super.tearDown();
          }
```

A continuación se muestra la página SessionDoc principal para una prueba `testSelfMadeMillionaireScenario`:

CREOLE Session

Generated: 13-Jul-2012 12:08:08

Session Type

- Recalculation strategy: curam.creole.execution.session.RecalculationsProhibited
- Data storage: curam.creole.storage.inmemory.InMemoryDataStorage
- Rule object factory: curam.creole.execution.session.StronglyTypedRuleObjectFactory

Options

- "Used by" links included: true

Rule Objects (by Rule Set)

- FlexibleRetirementYearRuleSet

Figura 6. SessionDoc para la prueba `testSelfMadeMillionaireScenario`

Algunos detalles clave en esta página son:

- La fecha y hora en la que se ha creado el SessionDoc;
- Las estrategias con las que se ha creado la sesión;
- Una lista de conjuntos de reglas cuyos objetos de regla han sido capturados en el SessionDoc y
- Indicar si se incluyen enlaces "utilizado por".

Al pulsar en el enlace para el único conjunto de reglas `FlexibleRetirementYearRuleSet`, se muestran los objetos de regla:

FlexibleRetirementYearRuleSet

Generated: 13-Jul-2012 12:08:08

External rule objects

Details	Type	Description	Action
details	FlexibleRetirementYearRuleSet.FlexibleRetirementYear	Undescribed instance of rule class 'FlexibleRetirementYear', id '1'	Created during this session

Internal rule objects

Details	Type	Description	Action

Figura 7. Objetos de regla para el conjunto de reglas `FlexibleRetirementYearRuleSet`

Esta página muestra:

- Los objetos de regla "externos" (programa de arranque) creados por el código de cliente durante esta sesión (sólo se ha creado uno en la prueba); y

- Los objetos de regla "internos" creados por las reglas durante esta sesión (no se ha calculado ninguno para esta prueba).

Al pulsar en el enlace de detalles para el único objeto de regla `FlexibleRetirementYear` se muestra su SessionDoc:

Created externally

Action during this session

Created during this session

Attributes

Name	Declared type	State	Value	Derivation	Depends on	Used by
ageAtRetirement	Number	CALCULATED	50	• If • retirementCause == / Then • "Lottery winner" / • 35 • "Self-made millionaire" / • 50 Otherwise / • 65	• retirementCause	• yearOfRetirement
description	Message	CALCULATED	Undescribed instance of rule class 'FlexibleRetirementYear', id '1'	• Default rule object description.	*None*	*None*
retirementCause	String	SPECIFIED	Self-made millionaire	• Specified externally.	*None*	• ageAtRetirement
yearOfBirth	Number	SPECIFIED	1980	• Specified externally.	*None*	• yearOfRetirement
yearOfRetirement	Number	CALCULATED	2030	• Arithmetic: ○ yearOfBirth ○ + ○ ageAtRetirement	• ageAtRetirement • yearOfBirth	*None*

Figura 8. SessionDoc para el objeto de regla `FlexibleRetirementYear`

En la parte superior (no se muestra) están los detalles de resumen para el objeto de regla y, a continuación, se lista cada atributo de regla en el objeto de regla, con los detalles siguientes:

- **Nombre**

  Nombre del atributo de regla;
- **Tipo declarado**

  Tipo del atributo de regla declarado en el conjunto de reglas. El valor de tiempo de ejecución real puede ser de un subtipo de este tipo declarado;
- **Estado**

  El estado del valor, es decir, si era:
  - **CALCULATED**

    Calculado por reglas;
  - **SPECIFIED**

    Especificado explícitamente por el código cliente, sustituyendo cualquier cálculo definido, o inicializado como parte de una expresión `create`;

    Nota: Antes de Cúram V6, se utilizaba el estado INITIALIZED. A partir de Cúram V6, se utiliza el estado SPECIFIED en su lugar.

- **NOT_YET_CALCULATED**

  No especificado explícitamente, no calculado durante la ejecución de reglas (porque ningún otro cálculo o prueba ha solicitado nunca el valor); o
- **ERROR**

  Se ha producido un error durante el cálculo del valor (consulte los registros de aplicación o la salida de consola para obtener detalles y la pila de cálculos del error).

- **Valor**

  Representación de visualización del valor. Si el valor no se ha calculado nunca (NOT_YET_CALCULATED) o es erróneo (ERROR), se visualizará "?". Si el valor es un objeto de regla, el valor se mostrará como un hiperenlace navegable para que pueda ver los detalles de dicho objeto de regla y
- **Derivación**

  La derivación de RuleDoc del atributo (sin enlaces). Para obtener más información sobre RuleDoc, consulte "RuleDoc" en la página 19.
- **Depende de**

  Enlaces a los atributos que se han utilizado para calcular este valor.
- **Utilizado por**

  (Opcional) Enlaces a los atributos que utilizaban este valor al calcular sus valores.

Consejo: La implementación de un atributo de regla `description` en cada clase de regla puede hacer que el atributo SessionDoc sea más fácil de entender. Consulte "El atributo de regla `description`" en la página 138 para obtener más detalles.

Si está ejecutando SessionDoc en una base de datos grande, puede ser útil suprimir la salida de los enlaces "utilizado por", porque la inclusión de enlaces de este tipo puede hacer que SessionDoc produzca un gran número de objetos de regla. Para suprimir la salida de los enlaces "utilizado por", utilice `curam.creole.execution.session.SessionDoc.write(File, boolean)`, pasando *false* como el segundo parámetro.

En el caso de objetos de regla que están almacenados en tablas de base de datos de CER, puede crear SessionDoc para estos objetos de regla almacenados ejecutando la clase `curam.creole.util.DumpOutRuleObjects`, donde un único argumento es el nombre de un directorio en el que se debe crear el SessionDoc. El programa de utilidad DumpOutRuleObjects recupera todos los objetos de regla de las tablas de base de datos de CER y, por lo tanto, se "recuperará" la acción para cada objeto de regla externa. Los objetos de regla interna se crearán (porque no están almacenados) y por lo tanto mostrarán una acción de "creado".

Consejo: El programa de utilidad DumpOutRuleObjects puede ser una manera útil de "navegación" en los objetos de regla almacenados en las tablas de base de datos de CER y puede ser una ayuda de depuración útil cuando haya alcanzado el punto de integración de reglas CER en la aplicación en línea.

Puede examinar los objetos de regla para ver los valores de los atributos calculados en objetos de regla, junto con una visión técnica de cómo se llegó a cualquier resultado de cálculo.

Atributos de regla no utilizados

CER incluye soporte para informar de atributos de regla a los que no se hace referencia desde ningún otro cálculo del conjunto de reglas y, por lo tanto, son candidatos para la eliminación.

Para ejecutar el informe de atributos no utilizado de CER, ejecute el mandato siguiente:

```
build creole.report.unused.attributes
```

CER validará los conjuntos de reglas e informará de los atributos de regla no utilizados en la consola.

aviso: Tenga en cuenta que es perfectamente posible que un atributo de regla sea un atributo de regla de "nivel superior" al que sólo se hace referencia desde el código de cliente; estos atributos pueden ser tratados como "no utilizados" por este informe, pero los atributos de regla aparentemente no utilizados no deben eliminarse del conjunto de reglas a menos que esté seguro de que ningún código de cliente o pruebas dependen de ellos.

Consolidador de conjunto de reglas CER

En Cúram V5.2, CER le permitía dividir el conjunto de reglas en archivos más pequeños que podían ayudar al desarrollo simultáneo de conjuntos de reglas entre varios documentos.

Consulte “Sentencia `Include`” en la página 159 para obtener detalles sobre cómo dividir el conjunto de reglas.

Antes de cargar el conjunto de reglas CER en los datos, los scripts de creación de la aplicación (específicamente el destino **`build creole.consolidate.rulesets`**) lo "consolidarán" automáticamente en un archivo de conjunto de reglas individual.

Nota: El consolidador de conjunto de reglas CER solo contrae las sentencias `Include` que contienen una ubicación `RelativePath`.

Todos los demás tipos de sentencia `Include` se dejan intencionalmente sin modificar en la salida consolidada.

Ejemplo

Esto es un conjunto de reglas CER que incluye otro conjunto de reglas:

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_Include"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">

  <!-- Esta clase de regla se define directamente en este conjunto de reglas -->
  <Class name="Person">
    <Attribute name="firstName">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>
  </Class>
```

```
  <!-- Incluir un conjunto de reglas definido en otro archivo.

       Cuando se ensamblan en un solo conjunto de reglas, los
       nombres de todas las clases de regla deben ser exclusivos. -->
  <Include>
    <RelativePath value="./HelloWorld.xml"/>
  </Include>

</RuleSet>
```

Y aquí se muestra el mismo conjunto de reglas después de la consolidación:

```
<?xml version="1.0" encoding="UTF-8"?><RuleSet
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 name="Example_Include" xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">

  <!-- Esta clase de regla se define directamente en este conjunto de reglas -->
  <Class name="Person">
    <Attribute name="firstName">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>
  </Class>

  <!-- Incluir un conjunto de reglas definido en otro archivo.

       Cuando se ensamblan en un solo conjunto de reglas, los
       nombres de todas las clases de regla deben ser exclusivos. -->

<!-- Iniciar inclusión de ./HelloWorld.xml-->
<Class name="HelloWorld">

    <Attribute name="greeting">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <String value="Hello, world!"/>
      </derivation>
    </Attribute>

  </Class>
<!--Finalizar inclusión de ./HelloWorld.xml-->


</RuleSet>
```

Manejo de datos

Una descripción de cómo CER maneja los datos. CER realiza cálculos en datos a fin de obtener resultados que sean ellos mismos datos. Desde Cúram V6, CER soporta el almacenamiento de los datos de objeto de regla en la base de datos y en instantáneas.

Sesiones de CER

Los principales elementos de datos utilizados con CER son:

- **Objetos de regla**

Un objeto de regla es una instancia de una clase de regla de un conjunto de reglas CER; por ejemplo, el objeto de regla para la persona "Juan Herrero" y

- **Valores de atributo**

  Un valor de atributo es el valor de un atributo de regla CER en un objeto de regla determinada; por ejemplo la fecha de nacimiento de Juan Herrero.

Toda la interacción con los objetos de regla CER y los valores de atributo se produce en una sesión CER. Estas interacciones incluyen la creación, recuperación y/o eliminación de objetos de regla CER y cualquier cálculo o recálculo de atributos en los objetos de regla.

Cada sesión CER se crea utilizando la clase `curam.creole.execution.session.Session_Factory`.

Cuando se crea una sesión CER, se debe especificar lo siguiente:

- **Estrategia de recálculo**

  La estrategia para manejar una solicitud para volver a calcular un valor de atributo CER en unasesión de CER

  Nota: La posibilidad de que CER efectúe recálculos directamente la reemplaza ahora el Gestor de dependencias (consulte “Gestor de dependencias” en la página 80). La interfaz de estrategia de recálculo de CER y las implementaciones se proporcionan sólo por compatibilidad con versiones anteriores.

  .

  La estrategia para manejar una solicitud para volver a calcular un valor de atributo CER; por ejemplo si se debe volver a calcular inmediatamente, aplazar el recálculo a otra transacción o no permitir el recálculo en absoluto.
- **Almacenamiento de datos**

  El uso del mecanismo de almacenamiento para los objetos de regla persistentes; por ejemplo sólo en la memoria interna (que se perderá cuando la sesión queda fuera del ámbito), el almacenamiento de base de datos o el almacenamiento de instantáneas.

  Nota:

  Desde Cúram V6, CER ofrece una selección de implementaciones de almacenamiento de datos. Estas implementaciones de almacenamiento de datos afectan si los objetos de reglas, creados o modificados en una transacción, se pueden recuperar o cambiar en otras transacciones.

  Lo que es más importante, la opción de almacenamiento de datos *no* afecta la semántica de las expresiones de regla. Esto significa que puede utilizar un almacenamiento de datos ligero (en memoria) para las pruebas de JUnit (para que los objetos de regla permanezcan entre transacciones), *sin* ninguna diferencia en los cálculos subyacentes.

A su vez, las implementaciones de almacenamiento de datos debe especificar una fábrica de objetos de regla a utilizar. Esta fábrica controla si los objetos de regla se crean de forma de tipo fuerte o de manera sólo interpretada.

La aplicación incluye estas implementaciones:

- Estrategia de recálculo:
  - **curam.creole.execution.session.RecalculationsProhibited**

Genera un error si se intenta cualquier recálculo con la sesión CER.

– **curam.core.sl.infrastructure.propagator.impl.ImmediateRecalculationStrategy**

  Realiza recálculos inmediatamente (de forma síncrona, en la misma transacción de base de datos).

– **curam.core.sl.infrastructure.propagator.impl.DeferredRecalculationStrategy**

  Aplaza los recálculos (para valores de atributo almacenados) a otra transacción de base de datos.

  Aplaza los recálculos (para valores de atributo almacenados) a otra transacción de base de datos.

• Almacenamiento de datos:

– **curam.creole.storage.inmemory.InMemoryDataStorage**

  Conserva los objetos de regla en memoria sólo. Los objetos de regla de este almacenamiento de datos sólo están disponibles mientras el almacenamiento de datos está en ámbito - normalmente sólo para una única transacción de base de datos.

– **curam.creole.storage.database.DatabaseDataStorage**

  Nota: Como optimización, sólo los objetos de reglas externos y sus valores de atributo se recuperan a partir de datos en la base de datos de Cúram. Los objetos de reglas internos y sus atributos, por su naturaleza, se pueden volver a calcular de forma fiable posteriormente, mientras que los objetos de regla externos normalmente contienen datos de orígenes externos y, de esta forma, no se pueden volver a calcular.

  Recupera objetos de regla externos de:

– **curam.creole.execution.session.RuleObjectsSnapshot.SnapshotDataStorage**

  Crea un documento XML que contiene detalles de un conjunto de objetos de regla implicados en las dependencias de uno o varios cálculos de atributo. Proporciona un "seguimiento de auditoría" de los datos utilizados finalmente en ese cálculo. Normalmente, el documento XML se puede almacenar en una tabla de base de datos para que la instantánea de objetos de regla se pueda consultar (pero no modificar) mediante una transacción de bases de datos posterior.

  - un conversor de objetos de regla registrado con el almacenamiento de datos de base de datos. Cada conversor de objeto de regla designa las clases de reglas que maneja y cuando se invoca al objeto de regla, el conversor leerá las tablas empresariales subyacentes para obtener los datos apropiados y llenará los objetos de regla en la memoria y los devolverá al almacenamiento de datos de base de datos; o
  - Las propias tablas de base de datos de CER para almacenar objetos de regla, si no hay ningún conversor de objeto de regla registrado para manejar la clase de regla solicitada.

  Nota: Tenga en cuenta que cada conversor de objeto de regla está autorizado para imponer límites en su soporte para expresiones de “`readall`” en la página 216 y “`readall`” en la página 216. Por ejemplo, es posible que algunos conversores de objeto de regla no soporten la ejecución de un “`readall`” en la página 216 (sin un `match` anidado), o coloquen restricciones en qué atributos de regla se pueden mencionar en el valor `retrievedattribute` de `match`. Cualquier violación de las limitaciones del conversor de objeto de regla hará que se genere una excepción en el tiempo de ejecución. Deberá asegurarse de que las pruebas del conjunto de reglas incluyen la lógica que invoca los conversores de objeto de regla (es decir, que se ejecuta en el almacenamiento

de datos de base de datos), a diferencia de la mayoría de las pruebas de la lógica de las reglas que utilizarán el almacenamiento de datos de la memoria interna (y que de estar no invocan los conversores de objeto de regla). Consulte la documentación de cada implementación de conversor de objeto de regla para conocer los límites que impone en el soporte para "readall" en la página 216.

Están disponibles objetos de regla externos para que las transacciones de base de datos posteriores los recuperen y manipulen.

– **curam.creole.storage.hybrid.HybridDataStorage**

Combina aspectos de comportamiento de InMemoryDataStorage y DatabaseDataStorage. *Reservado sólo para el uso interno de Cúram.*

- Fábrica de objetos de regla:
  – **curam.creole.execution.session.StronglyTypedRuleObjectFactory**

  Crea y recupera objetos de regla como instancias de clases Java generadas por el generador de código de pruebas de CER (consulte "Generador de códigos de prueba de CER" en la página 14).

  Nota: No se debe utilizar en el código de producción.

  – **curam.creole.execution.session.InterpretedRuleObjectFactory**

  Crea y recupera objetos de regla en una forma totalmente interpreta (y por lo tanto dinámica). (Consulte "Intérprete de conjunto de reglas" en la página 12).

PRECAUCIÓN:
En general, no se debe utilizar más de una sesión CER dentro de una transacción de base de datos. Las copias en memoria de objetos de regla en una sesión CER son independientes de las copias en memoria de los objetos de regla en todas las demás sesiones de CER.

No se garantiza el comportamiento si más de una sesión CER (en la misma transacción) intenta recuperar o consultar el mismo objeto de regla de la base de datos.

- **Pruebas de unidad**

  Utilice InMemoryDataStorage (para rapidez de ejecución) con StronglyTypedRuleObjectFactory (para que se puedan utilizar en las pruebas las clases Java generadas) y RecalculationsProhibited (para que las pruebas no cambien por accidente los datos durante una parte del proceso).

- **Lógica de producción con conjuntos de reglas dinámicos**

  Utilice DatabaseDataStorage (para que los objetos de regla estén disponibles a lo largo de las transacciones) con un InterpretedRuleObjectFactory (para que los conjuntos de reglas sean totalmente dinámicos) y RecalculationsProhibited y utilice las características proporcionadas por el Gestor de dependencias (consulte "Gestor de dependencias" en la página 80) para realizar las solicitudes para recalcular los valores CER en una sesión de CER nueva (e independiente).

- **Instantáneas**

  Utilice SnapshotDataStorage (para que los objetos de regla se lean de un documento XML no modificable) con un InterpretedRuleObjectFactory (para que los conjuntos de regla sean totalmente dinámicos) y RecalculationsProhibited (las instantáneas no soportan cambios).

Objetos de regla externa e interna

CER permite la creación de objetos de regla de estas formas diferentes:

- **Externo**

Objetos de regla creados por clientes de CER, fuera del contexto de cualquier cálculo. Los objetos de regla externa tienden a representar los conceptos del mundo real o de agencia, por ejemplo una persona o un caso.

- **Interno**

  Objetos de regla creados por las reglas CER (o dentro del código Java llamado por las reglas CER). Los objetos de regla interna tienden a ser "calculadoras" o datos derivados para un paso provisional en una cadena de cálculos compleja.

La distinción entre objetos de regla externa e interna se explica a continuación y afecta a:

- si un objeto de regla se puede recuperar utilizando la expresión "readall" en la página 216 y
- si un objeto de regla se puede almacenar en la base de datos para recuperarse en transacciones posteriores.

Objetos de regla externa

CER permite al código de cliente formular preguntas de un objeto de regla (y CER ejecutará las reglas para proporcionar las respuestas a estas preguntas).

Sin embargo, para que el código de cliente formule una pregunta de un objeto de regla, ese objeto de regla debe ser conocido en el código de cliente y CER; por eso, la sesión de reglas CER debe hacer que el código de cliente cree o recupere al menos un objeto de regla de programa de arranque ("bootstrap"). Este código de cliente puede ser código de prueba o código que integra CER con una aplicación.

Un objeto de regla externa es el punto de partida para que el código de cliente haga preguntas; sin embargo, la respuesta a dicha pregunta puede proporcionar un objeto de regla o una lista de objetos de regla que se han creado a partir de las reglas o se han recuperado de otros objetos de regla externa.

Importante: Una vez que han empezado los cálculos, la estrategia `RecalculationsProhibited` impide la creación de más objetos de regla que invaliden los cálculos de "readall" en la página 216 calculados anteriormente.

Para evitar errores de este tipo, debe estructurar el código de cliente o las pruebas para que la creación de todos los objetos de regla de prueba se produzca *antes* de cualquier cálculo (es decir antes de la ejecución de `getValue`).

Ejemplo: A continuación se muestra un conjunto de reglas de ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_externalRuleObjects"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <!-- Estos atributos se deben especificar en el momento de la creación -->
    <Initialization>
      <Attribute name="firstName">
        <type>
          <javaclass name="String"/>
        </type>
      </Attribute>

      <Attribute name="lastName">
        <type>
          <javaclass name="String"/>
        </type>
```

```
      </Attribute>
    </Initialization>

    <Attribute name="incomes">
      <type>
        <javaclass name="List">
          <ruleclass name="Income"/>
        </javaclass>
      </type>
      <derivation>
        <!-- Leer todos los objetos de regla de
              tipo "Income" -->
        <readall ruleclass="Income"/>
      </derivation>
    </Attribute>

  </Class>

  <Class name="Income">
    <Attribute name="amount">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

En el conjunto de reglas anterior, la expresión "readall" en la página 216 se utiliza para recuperar todas las instancia de la clase de regla Income.

Para crear un objeto de regla externa, el código de cliente o las pruebas deben especificar la sesión cuando se crea el objeto de regla:

```
package curam.creole.example;

import junit.framework.TestCase;
import curam.creole.calculator.CREOLETestHelper;
import curam.creole.execution.RuleObject;
import curam.creole.execution.session.InterpretedRuleObjectFactory;
import curam.creole.execution.session.RecalculationsProhibited;
import curam.creole.execution.session.Session;
import curam.creole.execution.session.Session_Factory;
import
 curam.creole.execution.session.StronglyTypedRuleObjectFactory;
import curam.creole.parser.RuleSetXmlReader;
import
 curam.creole.ruleclass.Example_externalRuleObjects.impl.Income;
import
 curam.creole.ruleclass.Example_externalRuleObjects.impl.Income_Factory;
import
 curam.creole.ruleclass.Example_externalRuleObjects.impl.Person;
import
 curam.creole.ruleclass.Example_externalRuleObjects.impl.Person_Factory;
import curam.creole.ruleitem.RuleSet;
import curam.creole.storage.inmemory.InMemoryDataStorage;

/**
 * Prueba objetos de regla externa creados directamente por el código de cliente.
 */
public class TestCreateExternalRuleObjects extends TestCase {

  /**
```

```
 * Ejemplo que muestra la creación de objetos de regla externa utilizando
 * código generado.
 */
public void testUsingGeneratedTestClasses() {

  final Session session =
      Session_Factory.getFactory().newInstance(
          new RecalculationsProhibited(),
          new InMemoryDataStorage(
              new StronglyTypedRuleObjectFactory()));

  /**
   * Tenga en cuenta que el compilador impone que se proporcione el tipo correcto de
   * argumentos de inicialización.
   */
  final Person person =
      Person_Factory.getFactory().newInstance(session, "Juan",
          "Herrero");
  CREOLETestHelper.assertEquals("Juan", person.firstName()
      .getValue());

  /**
   * Estos objetos los recuperará la expresión
   *
   * <readall ruleclass="Income"/>
   *
   * en el conjunto de reglas.
   */
  final Income income1 =
      Income_Factory.getFactory().newInstance(session);
  income1.amount().specifyValue(123);

  final Income income2 =
      Income_Factory.getFactory().newInstance(session);
  income2.amount().specifyValue(345);

}

/**
 * Ejemplo que muestra la creación de objetos de regla externa utilizando
 * el intérprete de conjunto de reglas CER.
 */
public void testUsingInterpreter() {

  /* leer en el conjunto de reglas */
  final RuleSet ruleSet = getRuleSet();

  /* iniciar una sesión interpretada */
  final Session session =
      Session_Factory.getFactory().newInstance(
          new RecalculationsProhibited(),
          new InMemoryDataStorage(
              new InterpretedRuleObjectFactory()));

  /**
   * Tenga en cuenta que el compilador no puede imponer que se proporcione el tipo correcto de
   * argumentos de inicialización se proporcionan - si son incorrectos
   * CER informará de un error de tiempo de ejecución.
   */
  final RuleObject person =
      session.createRuleObject(ruleSet.findClass("Person"),
          "Juan", "Herrero");
  CREOLETestHelper.assertEquals("Juan", person
      .getAttributeValue("firstName").getValue());

  /**
   * Estos objetos los recuperará la expresión
```

```
     *
     * <readall ruleclass="Income"/>
     *
     * en el conjunto de reglas.
     */
    final RuleObject income1 =
        session.createRuleObject(ruleSet.findClass("Income"));
    income1.getAttributeValue("amount").specifyValue(123);

    final RuleObject income2 =
        session.createRuleObject(ruleSet.findClass("Income"));
    income2.getAttributeValue("amount").specifyValue(345);
  }

  /**
   * Lee Example_externalRuleObjects del archivo de
   * origen XML.
   */
  private RuleSet getRuleSet() {

    /* La vía de acceso relativa al archivo de origen de conjunto de reglas */
    final String ruleSetRelativePath =
        "./rules/Example_externalRuleObjects.xml";

    /* leer en el origen de conjunto de reglas */
    final RuleSetXmlReader ruleSetXmlReader =
        new RuleSetXmlReader(ruleSetRelativePath);

    /* volcar los problemas */
    ruleSetXmlReader.validationProblemCollection().printProblems(
        System.err);

    /* fallar si hay errores en el conjunto de reglas */
    assertTrue(!ruleSetXmlReader.validationProblemCollection()
        .containsErrors());

    /* devolver el conjunto de reglas del lector */
    return ruleSetXmlReader.ruleSet();
  }

}
```

Cuándo utilizar objetos de regla externa: Debe crear objetos de regla externa para:

- objetos de regla de nivel superior o "programa de arranque" que siempre deben existir para que el código de cliente plantee preguntas significativas. Estos objetos de regla son normalmente singletons (es decir, la instancia única de la clase de regla particular durante la sesión) y
- objetos de regla que se crean basándose en algunos datos externos (por ejemplo persona o caso).

Objetos de regla interna

CER permite que las reglas creen objetos de regla nuevos como resultado o consecuencia de la realización de cálculos.

Para crear un objeto de regla, utilice la expresión "`create`" en la página 183, especificando los valores de argumentos de inicialización necesarios para la clase de regla y/o valores adicionales a especificar en el objeto de regla creada.

Importante: Los objetos de regla creados utilizando la expresión "`create`" en la página 183 *no se pueden* recuperar utilizando expresiones "`readall`" en la página 216, porque CER no puede garantizar que todos los objetos de regla interna se

hayan creado o se vayan a crear (en función de si se encuentra una expresión "create" en la página 183 en la vía de acceso de ejecución para un cálculo).

Ejemplo: A continuación se muestra un conjunto de reglas CER de ejemplo que utiliza la expresión "create" en la página 183 para crear condicionalmente objetos de regla desde reglas:

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_internalRuleObjects"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="age">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- Utiliza <create> para obtener un nuevo objeto de regla.

         Otros cálculos (en este u otros objetos de regla) pueden
         acceder a este objeto de regla recién creado haciendo referencia a
         este atributo, es decir

         <reference attribute="minorAgeRangeTest"/> .

         El objeto de regla creado NO PUEDE recuperarse utilizando una
         expresión <readall>.
     -->
    <Attribute name="minorAgeRangeTest">
      <type>
        <ruleclass name="AgeRangeTest"/>
      </type>
      <derivation>
        <!-- Crear una prueba de rango de edad que comprueba si esta
             persona tiene entre 0 y 17 años inclusive (es decir
             menos de 18 años).
          -->
        <create ruleclass="AgeRangeTest">
          <this/>
          <Number value="0"/>
          <Number value="17"/>
        </create>
      </derivation>

    </Attribute>

    <!-- Utiliza la comprobación de rango de edad para determinar si esta persona
         es un menor. -->
    <Attribute name="isMinor">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <reference attribute="isPersonInAgeRange">
          <reference attribute="minorAgeRangeTest"/>
        </reference>
      </derivation>
    </Attribute>

    <!-- Utiliza <create> para obtener un objeto de regla nuevo, dentro
         de otra expresión.
```

```
      Dado que el objeto de regla nuevo se crea de "forma anónima",
      no está disponible a ningún otro objeto de regla (pero
      se seguirá mostrando como creado ("created") en cualquier SessionDoc.
  -->
  <Attribute name="isOfWorkingAge">
    <type>
      <javaclass name="Boolean"/>
    </type>
    <derivation>
      <!-- Crear una prueba de rango de edad que comprueba si esta
           persona está legalmente autorizada a trabajar (es decir, tiene
           como mínimo 16 años y menos de 65) y, a continuación, comprobar
           si la prueba pasa. -->
      <reference attribute="isPersonInAgeRange">
        <create ruleclass="AgeRangeTest">
          <this/>
          <Number value="16"/>
          <Number value="64"/>
        </create>
      </reference>
    </derivation>

  </Attribute>

</Class>

<!-- Una prueba genérica que comprueba si la
     edad de la persona está dentro de un rango
     especificado (inclusive). -->
<Class name="AgeRangeTest">
  <Initialization>
    <Attribute name="person">
      <type>
        <ruleclass name="Person"/>
      </type>
    </Attribute>
    <Attribute name="minimumAge">
      <type>
        <javaclass name="Number"/>
      </type>
    </Attribute>
    <Attribute name="maximumAge">
      <type>
        <javaclass name="Number"/>
      </type>
    </Attribute>
  </Initialization>

  <Attribute name="isPersonInAgeRange">
    <type>
      <javaclass name="Boolean"/>
    </type>
    <derivation>
      <all>
        <fixedlist>
          <listof>
            <javaclass name="Boolean"/>
          </listof>
          <members>
            <compare comparison=">=">
              <reference attribute="age">
                <reference attribute="person"/>
              </reference>
              <reference attribute="minimumAge"/>
            </compare>
            <compare comparison="&lt;=">
```

```
              <reference attribute="age">
                <reference attribute="person"/>
              </reference>
              <reference attribute="maximumAge"/>
            </compare>
          </members>
        </fixedlist>
      </all>
    </derivation>
  </Attribute>

</Class>

</RuleSet>
```

Nota: Como para todas las expresiones CER, las expresiones “create” en la página 183 sólo se calcularán si se solicitan, por ejemplo el objeto de regla *minorAgeRangeTest* sólo se creará si su valor o el valor de *isMinor* se solicita mediante código de cliente u otro cálculo.

En el ejemplo anterior, *isOfWorkingAge* utiliza la técnica de crear un objeto de regla de "forma anónima" envolviendo la creación de un objeto de regla en una expresión que hace referencia a algún atributo en el objeto de regla recién creado. Estos objetos de regla no están disponibles para otros cálculos pero siguen mostrándose en cualquier SessionDoc generado.

Un objeto de regla anónimo puede ser útil cuando necesite acceder a los atributos de regla en un objeto de regla creado, pero no es necesario que el objeto de regla creado esté disponible para cualquier otro cálculo.

Agrupación de objetos de regla interno

Desde Cúram V6, CER mantiene una "agrupación" de objetos de regla internos que se han creado durante una sesión.

La agrupación de la sesión se consulta siempre que se evalúa una expresión “create” en la página 183. Si ya se ha creado un objeto de regla con los mismos parámetros especificados y/o de inicialización, se volverá a utilizar desde la agrupación en lugar de crearse un objeto de regla nuevo.

Este enfoque de agrupación mejora la eficiencia en la situación donde muchas sentencias “create” en la página 183 intentan crear objetos de regla "idénticos". El uso de un solo objeto de regla significa que los atributos calculados en el único objeto de regla se calculan como máximo una vez, en lugar de producirse cálculos idénticos en muchos objetos de regla "idénticos".

Se garantiza que la reutilización de los objetos de regla agrupados es segura, porque los principios básicos de CER aseguran que cualquier cálculo depende sólo de sus entradas y, por consiguiente, entradas idénticas garantizan salidas idénticas.

Cuándo utilizar objetos de regla interna: Debe utilizar esta modalidad para crear objetos de regla condicionalmente de acuerdo con las reglas o cuando se calculan los valores de atributo de inicialización desde otras reglas o cuando el objeto de regla es sólo un paso provisional en un cálculo.

Para obtener un cálculo muy complejo, debe esperar a tener un objeto de regla externa que contenga un atributo para el resultado del cálculo, varios objetos de

regla externa para los datos de entrada de fuera de las reglas y posiblemente un número muy grande de objetos de regla interna para los pasos de cálculo intermedios.

Si tiene objetos de regla que existirán siempre y/o necesitan que las expresiones "readall" en la página 216 accedan a ellos, en lugar de ello considere la posibilidad de crear objetos de regla externa directamente en su código.

Manejo de tipos de datos

Cada atributo y expresión en un conjunto de reglas CER devuelve una parte de los datos (cuando se solicitan). CER soporta un conjunto flexible de tipos de datos, que se debe especificar en cada atributo y algunas expresiones del conjunto de reglas.

Tipos de datos soportados

CER incluye soporte para los siguientes tipos de datos:

- Clases de regla;
- Clases Java y
- Tablas de código de aplicación.

Clases de regla: Cualquier clase de regla CER definida en el conjunto de reglas puede utilizarse como un tipo de datos en el mismo conjunto de reglas.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_ruleclassDataType"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="firstName">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="favoritePet">
      <type>
        <!-- El tipo de este atributo es una clase de regla
             definida en otro lugar de este conjunto de reglas.        -->
        <ruleclass name="Pet"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

  </Class>

  <Class name="Pet">

    <Attribute name="name">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>
```

```
  </Class>

</RuleSet>
```

Herencia

CER soporta una herencia de implementación simple para clases de regla.

Una clase de regla puede especificar opcionalmente una declaración *extends* para subclasificar otra clase de regla en el mismo conjunto de reglas.

Una clase de subregla hereda los atributos de regla calculados de todas sus clases ancestros y opcionalmente puede alterar temporalmente cualquiera de estos atributos para proporcionar reglas de cálculo de derivación diferentes.

Una clase de subregla también hereda los atributos de regla inicializados de todas las clases ancestro y cualquier expresión "`create`" en la página 183 para la clase de subregla debe especificar el valor de los atributos inicializados para todas las clases de regla ancestro de la clase de subregla *antes que* cualquier declarado en la propia clase de subregla.

CER permite declarar un atributo "`abstract`" en la página 164. Cada clase de regla que define o hereda (pero no altera temporalmente) un atributo abstracto debe declararse a sí misma abstracta. Una clase abstracta no puede utilizarse en una expresión "`create`" en la página 183.

CER permitirá que una instancia de objeto de regla de una clase de regla se devuelva siempre que se espere una de sus clases de regla ancestro.

El validador de conjunto de reglas CER informará de un error si una expresión del conjunto de reglas intenta devolver un valor incompatible:

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_ruleclassInheritance"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">


  <!-- El validador de conjunto de reglas CER insistirá en que esta
       clase de regla se marque como abstracta, porque contiene
       un atributo de regla abstracto. -->
  <Class name="Resource" abstract="true">
    <Initialization>
      <!-- Siempre que se crea un objeto de regla de recurso,
           se debe inciializar su propietario.

           Puesto que el recurso es abstracto, no se puede
           utilizar en una expresión <create>, sólo se
           pueden utilizar subclases concretas. -->
      <Attribute name="owner">
        <type>
          <ruleclass name="Person"/>
        </type>
      </Attribute>
    </Initialization>


    <!-- El valor monetario del recurso. -->
    <Attribute name="value">
      <type>
```

```
      <javaclass name="Number"/>
    </type>
    <derivation>
      <!-- Cada recurso tiene un importe, pero
           se calcula de una manera específica de subclase. -->
      <abstract/>
    </derivation>
  </Attribute>
</Class>

<!-- Un edificio es un tipo de recurso. -->
<Class name="Building" extends="Resource">
  <!-- La dirección física del edificio,
       por ejemplo Granvía 123.

       Se debe especificar el valor de dirección
       además del atributo de regla de propietario
       heredado, que es un
       atributo inicializado en la clase
       de super-regla. -->
  <Initialization>
    <Attribute name="address">
      <type>
        <javaclass name="String"/>
      </type>
    </Attribute>
  </Initialization>


  <!-- Building es una clase concreta
       y, por lo tanto, el validador de
       conjunto de reglas CER insistirá en que esta
       clase herede o declare un cálculo
       para todos los atributos de regla abstractos heredados. -->
  <Attribute name="value">
    <type>
      <javaclass name="Number"/>
    </type>
    <derivation>
      <arithmetic operation="-">
        <reference attribute="purchasePrice"> </reference>
        <reference attribute="outstandingMortgageAmount"/>
      </arithmetic>

    </derivation>
  </Attribute>

  <!-- El precio pagado originalmente por el edificio. -->
  <Attribute name="purchasePrice">
    <type>
      <javaclass name="Number"/>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

  <!-- El importe de préstamos o hipotecas pendientes para
       este edificio. -->
  <Attribute name="outstandingMortgageAmount">
    <type>
      <javaclass name="Number"/>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>
```

```
  </Class>

  <Class name="Vehicle" extends="Resource">
    <Initialization>
      <Attribute name="registrationPlate">
        <type>
          <javaclass name="String"/>
        </type>
      </Attribute>
    </Initialization>

    <Attribute name="value">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <!-- El valor de este tipo de recurso se
             especifica directamente, en lugar de calcularse.->
        <specified/>

      </derivation>
    </Attribute>

  </Class>

  <Class name="Person">

    <!-- Un atributo de ejemplo que muestra cómo se heredan los
         atributos inicializados. -->
    <Attribute name="sampleBuilding">
      <type>
        <ruleclass name="Building"/>
      </type>
      <derivation>
        <create ruleclass="Building">
          <!-- el primer atributo de regla inicializado
               se hereda de Resource.

               Establecer que esta persona sea el propietario -->
          <this/>
          <!-- El segundo atributo de regla inicializado
            se especifica directamente en Building.

            Establecer la dirección del edificio. -->
          <String value="Granvía 123"/>
        </create>
      </derivation>
    </Attribute>

    <!-- un atributo de ejemplo que muestra cómo un edificio puede ser
         devuelto como un recurso (porque un edificio *ES* un
         recurso -->
    <Attribute name="sampleResource">
      <type>
        <ruleclass name="Resource"/>
      </type>
      <derivation>
        <reference attribute="sampleBuilding"/>
      </derivation>
    </Attribute>


  </Class>

</RuleSet>
```

Clase de regla raíz

Si una clase de regla no especifica otra clase de regla para ampliarse, la clase de regla amplía automáticamente la clase de regla "raíz" de CER, que contiene un solo atributo de regla `description`.

El atributo de regla `description` proporciona una descripción localizable de la instancia de objeto de regla. Las clases de reglas pueden alterar temporalmente la derivación del cálculo de regla de `description` para las instancias de objeto de regla.

Finalmente cada clase de regla hereda de la clase de regla raíz (y, por lo tanto contiene un atributo de regla `description`), de forma similar a cómo todas las clases Java heredan finalmente de `java.lang.Object`.

La implementación predeterminada del atributo de regla `description`, proporcionado por la clase de regla raíz, utiliza la expresión "`defaultDescription`" en la página 192.

Clases Java: Cualquier clase Java en la vía de acceso de clases de la aplicación puede utilizarse como un tipo de datos en el conjunto de reglas CER.

PRECAUCIÓN:
Al almacenar objetos de regla en la base de datos, sólo se pueden utilizar los tipos de datos para los que existe un manejador de tipos registrado en CER.

CER incluye manejadores de tipo para la mayoría de los tipos de datos utilizados comúnmente.

Nombres de paquete

El nombre de una clase Java debe estar completo con el nombre de paquete, excepto para las clases de los paquetes siguientes:

- `java.lang.*` y
- `java.util.*`.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_javaclassDataType"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">
    <Attribute name="isMarried">
      <type>
        <!-- java.lang.Boolean no necesita que se
             especifique el paquete -->
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="dateOfBirth">
      <type>
        <!-- Nombre completo para una clase de Cúram -->
        <javaclass name="curam.util.type.Date"/>
      </type>
      <derivation>
        <specified/>
```

```
      </derivation>
    </Attribute>


  </Class>

</RuleSet>
```

Consejo: Los tipos Java primitivos como `boolean` no se pueden utilizar en CER; en su lugar, utilice los equivalentes de clase (por ejemplo `Boolean`).

Objetos inmutables

Un principio básico de CER es que cada valor, una vez calculado, no se puede cambiar.

Para cumplir con este principio, las clases Java que utilice deben ser *inmutables*.

aviso: Si utiliza una clase Java *mutable* como tipo de datos en el conjunto de reglas CER, debe asegurarse de que ningún código Java intenta modificar el valor de los objetos de esa clase Java. CER no puede garantizar la fiabilidad de los cálculos si los valores se están cambiando "por debajo".

Afortunadamente, hay una amplia gama de clases inmutables que se proporciona normalmente para la mayoría de los requisitos de tipo de datos. En general es posible que necesite examinar el JavaDoc de una clase Java para determinar si es inmutable.

A continuación se listan algunas clases inmutables útiles que probablemente serán suficientes para sus necesidades:

- `java.lang.String`;
- `java.lang.Boolean`;
- Implementaciones de `java.lang.Number`; en cualquier caso, CER convierte las instancias de `Number` a su propio formato numérico (respaldado por `java.math.BigDecimal`) antes de realizar cualquier aritmética o comparación;
- Implementaciones de `java.util.List` que *no* soportan las operaciones opcionales (consulte el JavaDoc para `List`);
- `curam.util.type.Date`;
- Implementaciones de `curam.creole.value.Message` y
- `curam.creole.value.CodeTableItem`.

Herencia

CER reconoce la jerarquía de herencia de clases e interfaces Java.

CER permitirá que un valor de una clase Java se devuelva en cualquier lugar en el que se espere una de las clases o interfaces Java de ancestro:

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_javaclassInheritance"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">
    <Attribute name="isMarried">
      <type>
        <javaclass name="Boolean"/>
      </type>
```

```
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="isMarriedAsObject">
      <type>
        <!-- Para simplificar el ejemplo, devolviendo este
             valor como un java.lang.Object (lo que no es
             probable que sea útil en un conjunto de
             reglas "real". -->
        <javaclass name="Object"/>
      </type>
      <derivation>
        <!-- Esto es correcto, porque un Booleano es (*IS*) un objeto. -->
        <reference attribute="isMarried"/>
      </derivation>
    </Attribute>

    <Attribute name="isMarriedAsString">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <!-- El validador de conjunto de reglas CER, informará del error
             siguiente (porque un Boolean no es (*IS NOT*) una serie):

             ERROR    Person.isMarriedAsString
             Example_javaclassInheritance.xml(28, 41)
             Child 'reference' returns 'java.lang.Boolean',
             but this item requires a 'java.lang.String'. -->
        <!-- <reference attribute="isMarried"/> -->

        <!-- (Declarar como especificado de forma que este ejemplo
              se cree de manera limpia) -->
        <specified/>

      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

Clases parametrizadas

Java 5 presentaba el soporte para clases parametrizadas y CER le permite utilizar clases Java parametrizadas en el conjunto de reglas.

Los parámetros para una clase parametrizada se listan simplemente en la declaración <javaclass>:

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_javaclassParameterized"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">
    <Attribute name="favoriteWords">
      <type>
        <!-- Una lista de series -->
        <javaclass name="List">
          <javaclass name="String"/>
        </javaclass>
      </type>
      <derivation>
```

```
    <specified/>
  </derivation>
</Attribute>

<Attribute name="luckyNumbers">
  <type>
    <!-- Una lista de números -->
    <javaclass name="List">
      <javaclass name="Number"/>
    </javaclass>
  </type>
  <derivation>
    <specified/>
  </derivation>
</Attribute>

<Attribute name="children">
  <!-- Una lista de objetos de regla de persona.

       Dado que java.util.List se puede parametrizar con
       cualquier objeto, podemos utilizar una clase de regla como parámetro.
  -->
  <type>
    <javaclass name="List">
      <ruleclass name="Person"/>
    </javaclass>
  </type>
  <derivation>
    <specified/>
  </derivation>
</Attribute>


<!-- Los perros que son propiedad de esta persona. -->
<Attribute name="dogs">
  <type>
    <javaclass name="List">
      <ruleclass name="Dog"/>
    </javaclass>
  </type>
  <derivation>
    <specified/>
  </derivation>
</Attribute>

<!-- Los gatos que son propiedad de esta persona. -->
<Attribute name="cats">
  <type>
    <javaclass name="List">
      <ruleclass name="Cat"/>
    </javaclass>
  </type>
  <derivation>
    <specified/>
  </derivation>
</Attribute>


<!-- Todas las mascotas que son propiedad de esta persona. -->
<Attribute name="pets">
  <type>
    <javaclass name="List">
      <ruleclass name="Pet"/>
    </javaclass>
  </type>
  <derivation>
    <joinlists>
```

```
        <fixedlist>
          <listof>
            <javaclass name="List">
              <ruleclass name="Pet"/>
            </javaclass>
          </listof>
          <members>
            <!-- todos los perros - los perros son un tipo de mascota -->
            <reference attribute="dogs"/>
            <!-- todos los gatos - los gatos son un tipo de mascota -->
            <reference attribute="cats"/>

            <!-- CER no permitirá "children" en esta
                 expresión; un niño no es una mascota independientemente
                 de que sea adorable o arañe
                 los muebles.  -->
            <!-- NO SE PUEDE UTILIZAR -->
            <!-- <reference attribute="children"/> -->
            <!-- NO SE PUEDE UTILIZAR -->

          </members>
        </fixedlist>

      </joinlists>
    </derivation>
  </Attribute>


</Class>

<Class abstract="true" name="Pet">

  <Attribute name="name">
    <type>
      <javaclass name="String"/>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

</Class>

<Class name="Dog" extends="Pet">

  <Attribute name="favoriteTrick">
    <type>
      <javaclass name="String"/>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

</Class>

<Class name="Cat" extends="Pet">

  <Attribute name="numberOfLives">
    <type>
      <javaclass name="Number"/>
    </type>
    <derivation>
      <!-- Todo el mundo sabe que los gatos
           tienen 9 vidas. -->
      <Number value="9"/>
    </derivation>
```

```
    </Attribute>

  </Class>

</RuleSet>
```

CER le permite utilizar cualquier tipo (incluidos los objetos de regla) para los parámetros que permiten `java.lang.Object`. CER aplicará "tipificación firme" aunque el parámetro (por ejemplo, una clase de regla) se define dinámicamente. CER también reconocerá la jerarquía de herencia de las clases de reglas al decidir si una clase parametrizada es asignable a otra.

Tablas de códigos: Se puede utilizar cualquier tabla de códigos de aplicación como tipo de datos en el conjunto de reglas CER.

Consejo: La tabla de códigos *no* necesita existir necesariamente en el tiempo de desarrollo; si un usuario administrativo utiliza la aplicación en línea para crear una nueva tabla de códigos, esa tabla de códigos se puede utilizar como tipo de datos en conjuntos de reglas CER definidos dinámicamente.

Para crear una instancia de una entrada de tabla de códigos (es decir, para hacer referencia a un elemento concreto de la tabla de códigos), utilice la expresión "Code" en la página 180.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_codetableentryDataType"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="gender">
      <type>
        <!-- El valor de este atributo será
             una entrada de la tabla de códigos "Gender"
             de Cúram. -->
        <codetableentry table="Gender"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="isMale">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <!-- Utilizar "Code" para crear un valor de codetableentry
        para la comparación. -->
        <equals>
          <reference attribute="gender"/>
          <Code table="Gender">
            <!-- El código de la tabla de códigos -->
            <String value="MALE"/>
          </Code>
        </equals>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

Dónde especificar tipos de datos

Cada "Atributo" en la página 161 (calculado e inicializado) debe especificar su tipo (`type`).

Para la mayoría de las expresiones, el `type` es fijo (por ejemplo la expresión "`all`" en la página 168 devuelve siempre un `booleano`) o se puede deducir (por ejemplo un "`reference`" en la página 221 devuelve el tipo declarado por el atributo (`Attribute`) de referencia).

Sin embargo, en el caso de algunas expresiones se debe especificar el tipo (`type`) explícitamente. Estas expresiones son:

- "`call`" en la página 175 y
- "`choose`" en la página 177.

Adicionalmente, la expresión "`fixedlist`" en la página 199 declara el tipo de elemento en la `Lista` devuelta en la sentencia `listof`.

Manejo de datos que cambian a lo largo del tiempo

Desde Cúram V6, CER soporta una característica potente denominada Líneas de tiempo. Una línea de tiempo CER es simplemente un valor que varía con el tiempo y es la simplicidad de este concepto que permite utilizar líneas de tiempo con un gran efecto en la aplicación.

¿Qué son datos de línea de tiempo?

Una línea de tiempo es una secuencia de valores de un tipo determinado, donde cada valor es efectivo a partir de una fecha determinada (hasta que se reemplaza por otro valor). Para cualquier fecha proporcionada, una línea de tiempo tiene un valor aplicable a dicha fecha.

Ejemplos de datos que se pueden modelar como líneas de tiempo de CER: Para presentar el concepto de línea de tiempo, se muestran a continuación algunos ejemplos de datos cotidianos que pueden variar a lo largo del tiempo:

- Los ingresos totales de una persona tendrán tendencia a variar a lo largo del tiempo a medida que la persona reciba aumentos de salario o cambie de empleo. Dado que los ingresos de una persona en un momento específico pueden representarse como un número, los ingresos de una persona que van variando a lo largo del tiempo se pueden representar como una línea de tiempo de <Número>;

  Nota: La notación utilizada en esta guía toma prestado intencionadamente esto de los valores genéricos de Java.
- Independientemente de los ingresos totales, el registro de empleo de una persona tenderá a variar a lo largo del tiempo a medida que la persona cambie de trabajo (o haya periodos de tiempo en los que la persona no tiene trabajo). Si en algún momento una persona tiene como máximo empleo *principal*, el historial del empleo principal de la persona puede representarse como una Línea de tiempo<Empleo>, donde Empleo es alguna clase de regla o tipo Java que contiene detalles de empleo, y durante los periodos en los que no tiene ningún empleo principal el valor de la línea de tiempo es un valor de marcador especial como `null` (para representar "sin empleo");
- Una persona puede ser propietaria de un activo que finalmente se desecha, por ejemplo una persona puede comprar y vender un coche. En cualquier fecha, la persona es propietaria del vehículo o no lo es, lo cual se puede modelar como un valor booleano. Con el tiempo, la condición de que el coche sea propiedad en

una fecha determinada se puede modelar como una Línea de tiempo<Booleano>. El valor de la línea de tiempo será `false` antes de la fecha de compra del coche y `true` a partir de la fecha de compra hasta e incluyendo la fecha de venta (o “hasta nuevo aviso” si no se sabe si se ha vendido el coche).

- Del mismo modo, una persona tiene una fecha de nacimiento y, finalmente, una fecha de defunción. Las personas que siguen vivas tienen registrada una fecha de defunción en blanco. En cualquier fecha, la persona está viva o muerta y, por lo tanto, el valor derivado para “?está viva la persona?” se puede modelar como un valor booleano. Con el tiempo, la condición de que la persona esté viva se puede modelar como una Línea de tiempo<Booleano>. El valor de la línea de tiempo será `false` antes de la fecha de nacimiento de la persona y será `true` a partir de la fecha de nacimiento de la persona hasta e incluyendo la fecha de defunción (o “hasta nuevo aviso” si la persona no tiene fecha de defunción).
- Un padre puede tener muchos hijos, nacidos en fechas diferentes. En una fecha determinada, el padre tiene una lista de hijos que están vivos en esa fecha, lo que se puede modelar como Lista<Persona>. Con el tiempo, la lista de hijos cambiará puesto que nacerán más hijos o los hijos llegarán a la edad madura (o con menos fortuna, morirán en la infancia), lo cual se puede modelar como Línea de tiempo<Lista<Persona>>

Los ejemplos anteriores se muestran en formato gráfico en la figura siguiente.

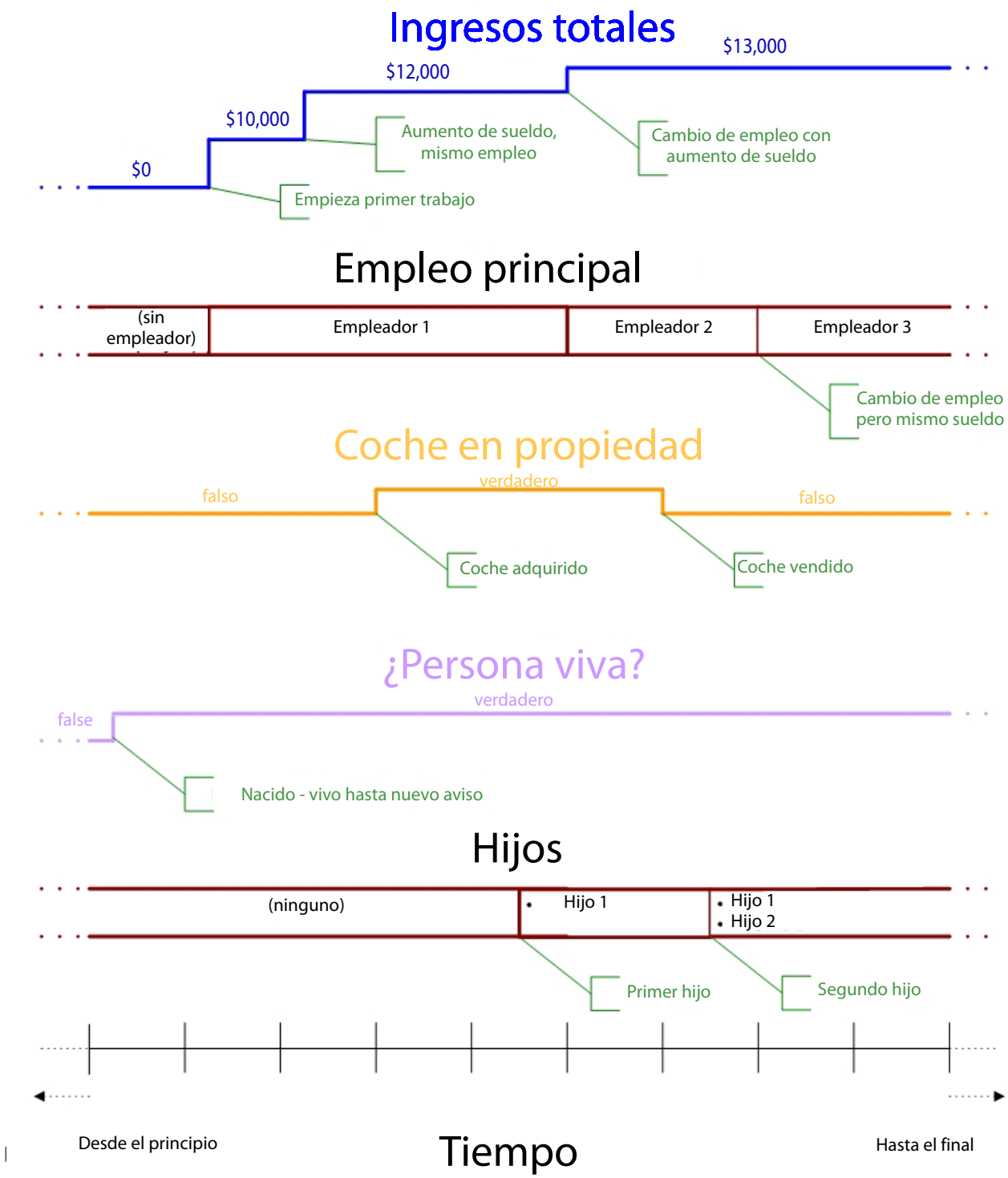

Figura 9. Ejemplos de datos de línea de tiempo

Ejemplos de datos que no son de línea de tiempo: Antes de continuar examinando las líneas de tiempo de manera más detallada, una nota de precaución sobre los datos que tienden a *no* ser adecuados para las líneas de tiempo.

Determinados tipos de datos no son adecuados para las líneas de tiempo, ya que los datos no varían con el tiempo. Son ejemplos comunes:

- **Identificadores exclusivos**

  Una de las características de un identificador exclusivo es que deliberadamente no varía a lo largo del tiempo, por ejemplo cada persona puede tener asignado un número de la seguridad social exclusivo. El número de la seguridad social se debe modelar como un número, no una línea de tiempo<Número>. Por el contrario, el nombre de una persona puede variar con el tiempo, por ejemplo debido al matrimonio o a un cambio de nombre por escritura unilateral, que es una de las razones por la que esta es una mala elección como identificador (además de la falta de exclusividad);
- **Fechas**

  Los datos del tipo de fecha no varían con el paso del tiempo. Por ejemplo, la fecha de nacimiento de una persona es una fecha concreta y por lo tanto no se debe modelar como fecha, ni como una Línea de tiempo<Fecha>. Las fechas pueden utilizarse para *construir* una línea de tiempo (por ejemplo, la fecha de nacimiento y la fecha de defunción de una persona se utilizarán para construir una Línea de tiempo<Booleano> para determinar si la persona está viva, pero las fechas en sí mismas no son líneas de tiempo).

  Nota: En Cúram, determinados fragmentos de datos como, por ejemplo, pruebas, pueden tener *dos* tipos de historial almacenados:
  - Un historial de *sucesiones* de datos relacionados con los cambios de circunstancias en el mundo real, es decir, donde se produce un suceso en el mundo real lo que significa que la representación del sistema de dichos datos está ahora anticuada, por ejemplo un cambio en los ingresos de una persona debido a un aumento salarial y
  - Un historial de *correcciones* en los datos mantenidos el sistema, donde el sistema tiene una representación incorrecta de las circunstancias del mundo real, por ejemplo se encuentra que la fecha de inicio de empleo de una persona se ha registrado correctamente.

  Por eso, para elementos de datos de tipo fecha, los datos se pueden *corregir* en el sistema, pero nunca *con éxito*. De este modo puede haber un *historial de correcciones* para el elemento de datos, pero este historial de correcciones (es decir las fechas en las que se ha creado el elemento de datos) raramente tiene interés al crear las reglas CER.

  Normalmente, los tipos de datos utilizados en las reglas CER deben modelar circunstancias del mundo real, en lugar de correcciones en la representación del sistema. Utilice la línea de tiempo donde esas circunstancias del mundo real puedan cambiar a lo largo del tiempo y no utilice la línea de tiempo donde los datos del mundo real no puedan variar con el tiempo.
- **Datos de punto en el tiempo**

  Algunos datos capturan intencionadamente datos que se aplican sólo a una fecha determinada. Por ejemplo, un elemento de datos para `surnameAtBirth` se debe modelar como una serie, no una línea de tiempo<Serie>. Un elemento de datos para `incomeAtRetirement` se debe modelar como un número, no como una Línea de tiempo<Número>

PRECAUCIÓN:
Al modelar datos es muy importante tener claro si el elemento de datos realmente varía a lo largo del tiempo. Utilice sólo la línea de tiempo para elementos de datos donde el valor puede variar con el tiempo.

Comparación de perspectivas de línea de tiempo y punto en el tiempo

CER maneja las líneas de tiempo de una manera natural, para que cuando diseñe reglas CER pueda conmutar mentalmente hacia atrás y hacia adelante entre una perspectiva de punto en el tiempo y una perspectiva de línea de tiempo.

En los apartados siguientes se describen estas perspectivas a modo de ejemplo. En primer lugar, veamos una perspectiva de punto en el tiempo, que no implica líneas de tiempo. A continuación, volveremos a visitar el ejemplo desde una perspectiva de línea de tiempo.

Perspectiva de punto en el tiempo: Supongamos que tenemos el requisito empresarial siguiente para una derivación:

regla: En cualquier fecha determinada, se considera que una persona es un *progenitor solo de un menor de edad* si, en esa fecha, la persona:

- *no está casada* y
- tiene un *dependiente que tiene menos de 16 años de edad*.

A partir de este simple requisito, es posible construir una simple tabla de decisión lógica para determinar si se considera que una persona es un progenitor solo de un menor de edad, *en una fecha determinada*:

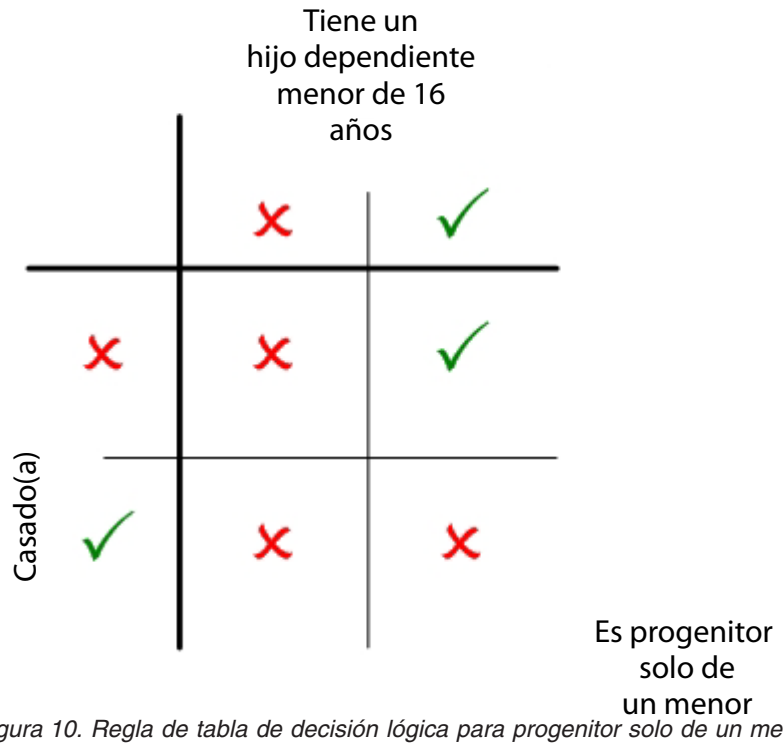

Figura 10. Regla de tabla de decisión lógica para progenitor solo de un menor de edad

Vamos a presentar un ejemplo de cambio real de circunstancias. El 1 de enero de 2001, María y Juan se casan. Juan tiene un hijo, Jaime, de un matrimonio anterior que terminó en divorcio el 30 de noviembre de 1998. Jaime nació el 1 de junio de 1990. El 30 de abril de 2004, lamentablemente Juan muere (y, por lo tanto, el matrimonio de María acaba en viudedad).

Podemos utilizar la tabla de decisión lógica anterior para determinar si cada una de las personas se considera como *progenitor solo de un menor de edad* en diversas fechas:

- El 1 de octubre de 1997 (para seleccionar una fecha al azar), María no es progenitor solo de un menor de edad porque en esa fecha no está casada, pero no tiene ningún dependiente;
- El 2 de octubre de 1997, María aún no es progenitor solo de un menor de edad, porque sus circunstancias no han cambiado desde el día anterior;
- El 30 de noviembre de 1998 Juan no es progenitor solo de un menor de edad , porque en esa fecha su hijo tenía menos de 16 años pero Juan aún estaba casado;
- El 1 de diciembre de 1998 Juan se convierte en progenitor solo de un menor de edad, porque en esa fecha su hijo aún tenía menos de 16 años pero Juan ya no estaba casado;
- En 1 de enero de 2001, María todavía no era progenitor solo de un menor de edad; aunque ella ahora tiene un dependiente menor de 16 años, está casada, de modo que por razones ligeramente diferentes a lo anterior, aún no es progenitor solo;
- El 1 de enero de 2001 Juan ya no es progenitor solo de un menor de edad; aunque aún tiene un dependientes menor de 16 años, está ahora se ha casado de nuevo;
- El 1 de mayo de 2004, María se convierte en progenitor solo de un menor de edad, porque su matrimonio ha finalizado debido a la muerte de Juan, pero Jaime todavía es su dependiente y es menor de 16 años;
- El 1 de junio de 2006, María deja de ser progenitor solo de un menor de edad, porque Jaime cumple 16 años
- El 1 de marzo de 2009 (de nuevo al azar) Jaime no es progenitor solo de un menor de edad porque aunque no está casado, no tiene dependientes.

Tenga en cuenta que tenemos que evaluar la tabla de decisión lógica para varias fechas a fin de crear una imagen que represente cuándo cada persona es progenitor solo de un menor de edad o no lo es. En cierta medida, tenemos que probar fechas que sospechamos que pueden ser "interesantes" o probar fechas al azar. Por ejemplo, sospechamos que la fecha de matrimonio de María puede ser interesante, pero resulta que su matrimonio con Juan *no* afecta su estado de progenitor solo de un menor de edad. Sin embargo, el estado de progenitor solo de un menor de edad de Juan *sí* cambia cuando se casa con María. No hemos pensado en probar si Juan era progenitor solo de un menor de dad en fechas anteriores al nacimiento de Jaime.

Perspectiva de línea de tiempo: Ahora vamos a volver a visitar la regla de ejemplo y las circunstancias desde una perspectiva de línea de tiempo.

En primer lugar, vamos a reformular ligeramente el requisito como se indica a continuación:

regla (reformulada): Se considera que una persona es un *progenitor solo de un menor de edad* siempre que la persona:

- *no está casada* y
- tiene un *dependiente que tiene menos de 16 años de edad*.

(Hemos eliminado las frases “En una fecha determinada” y “en esa fecha” y en su lugar utilizamos “siempre que”. Esta reformulación es sutil, pero puede ser clave para ayudar a hacer el cambio mental de pensar en puntos en el tiempo a pensar en datos que cambian con el tiempo.)

Ahora vamos dibujar líneas de tiempo (de tipo Línea de tiempo<Booleano> para el momento en que cada una de las personas:

- está casada y
- tiene un dependiente menor de 16 años de edad.

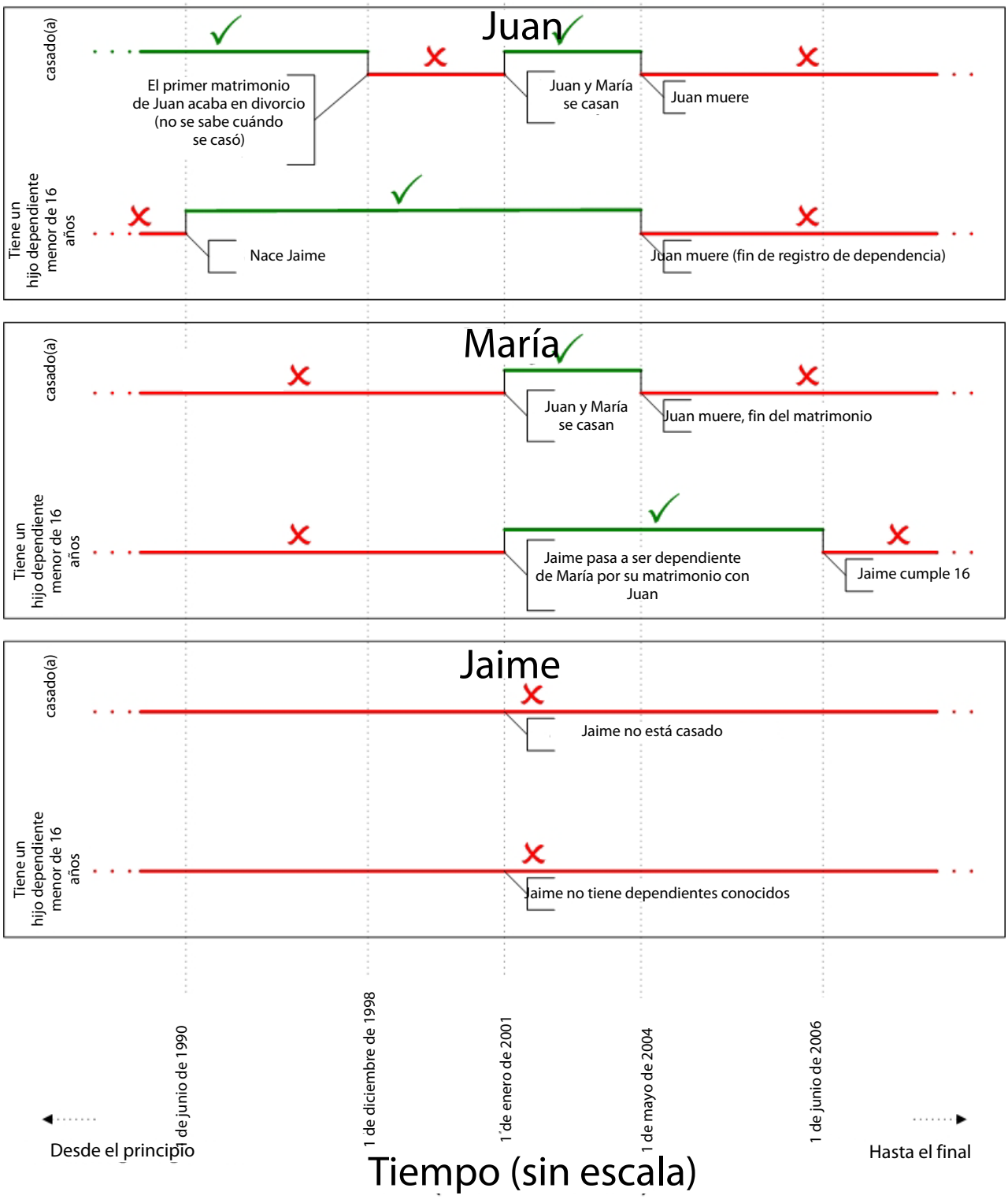

Figura 11. Líneas de tiempo para las circunstancias de Juan, María y Jaime

A partir de estas líneas de tiempo de las circunstancias de Juan, María y Jaime, vamos a dibujar líneas de tiempo nuevas para obtener cómo cambian sus estados de progenitor solo de menor de edad a lo largo del tiempo:

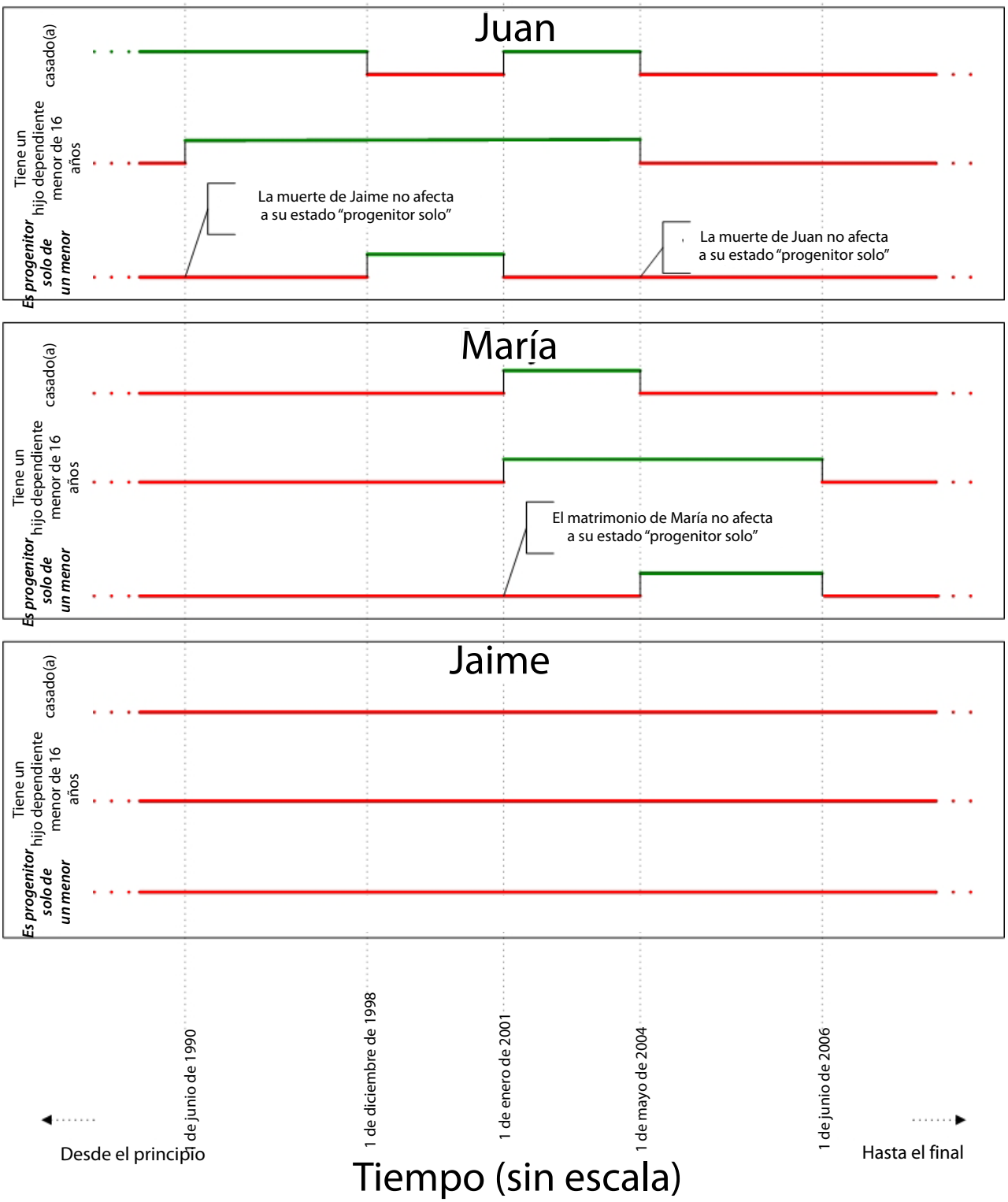

Figura 12. Líneas de tiempo para el estado de Juan, María y Jaime como progenitor solo de un menor de edad

Tenga en cuenta que podemos leer inmediatamente cómo cambia el estado de progenitor solo de un menor de edad de cada persona, sin tener que adivinar fechas que son de interés:

- Juan es progenitor solo de un menor de edad desde el 1 de diciembre de 1998 hasta el 31 de diciembre de 2000 inclusive;

- María es progenitor solo de un menor de edad desde el 1 de mayo de 2004 hasta el 30 de junio de 2006 inclusive y
- Jaime no es nunca un progenitor solo de un menor de edad.

Construcción de líneas de tiempo

Hemos visto cómo aplicar expresiones a los datos de línea de tiempo existentes previamente con el fin de calcular los datos que, a su vez, son una línea de tiempo.

En esta sección se describe en primer lugar cómo se crean datos de línea de tiempo.

Hay dos formas en las que se pueden crear líneas de tiempo:

- en el código Java, por parte de un cliente de CER o dentro del código Java invocado durante una llamada de CER y/o
- en las reglas CER, utilizando expresiones de CER que crean datos de línea de tiempo a partir de datos (no de línea de tiempo) de primitiva.

Construcción de líneas de tiempo en código Java: En Java, cada fragmento de datos de línea de tiempo es una instancia de la clase parametrizada `curam.creole.value.Timeline`. Para obtener detalles completos de esta clase, consulte el JavaDoc disponible en `EJBServer/components/CREOLEInfrastructure/doc` en una instalación de desarrollo de la aplicación.

Cada línea de tiempo contiene una colección de Intervalos, donde un intervalo es *valor* aplicable a partir de una determinada *fecha de inicio*. Se debe pasar una colección de intervalos adecuados al constructor de la línea de tiempo.

Por ejemplo, suponga que necesita crear una línea de tiempo<Número> con estos intervalos (recuerde que una línea de tiempo se extiende infinitamente en el pasado y el futuro):

- 0 hasta e incluyendo el 31 de diciembre de 2000;
- 10.000 desde el 1 de enero de 2001 hasta e incluyendo el 30 de noviembre de 2003 y
- 12.000 desde el 1 de diciembre de 2004 hasta nuevo aviso.

A continuación se muestra código Java de ejemplo para crear una línea de tiempo de esta clase:

```
package curam.creole.example;

import curam.creole.value.Interval;
import curam.creole.value.Timeline;
import curam.util.type.Date;

public class CreateTimeline {

  /**
   * Crea una línea de tiempo de número con estos valores de intervalo:
   * <ul>
   * <li>0 hasta e incluyendo el 31 de diciembre de 2000;</li>
   * <li>10.000 desde el 1 de enero de 2001 hasta e incluyendo el 30
   * de noviembre de 2003 y</li>
   * <li>12.000 desde el 1 de diciembre de 2004 hasta nuevo aviso.</li>
   * </ul>
   */
  public static Timeline<Number> createNumberTimeline() {

    return new Timeline<Number>(
```

```
    // primer intervalo, aplicación desde el "inicio del tiempo"
        new Interval<Number>(null, 0),

        // segundo intervalo
        new Interval<Number>(Date.fromISO8601("20010101"), 10000),

        // último intervalo (hasta nuevo aviso)
        new Interval<Number>(Date.fromISO8601("20041201"), 12000)

    );

  }
}
```

A continuación se muestra otro ejemplo de código de Java, que se puede llamar una expresión `call` CER, para calcular una línea de tiempo para la edad de una persona, hasta el 200 cumpleaños de la persona (recuerde que las líneas de tiempo de edad se deben limitar de manera artificial para que la línea de tiempo contenga un número finito de cambios de valor):

```
package curam.creole.example;

import java.util.ArrayList;
import java.util.Calendar;
import java.util.Collection;

import curam.creole.execution.session.Session;
import curam.creole.value.Interval;
import curam.creole.value.Timeline;
import curam.util.type.Date;

public class AgeTimeline {

  /**
   * Crea una línea de tiempo para la edad de una persona, artificialmente
   * limitada a 200 cumpleaños.
   * <p>
   * Se puede invocar desde las reglas CER a través de una expresión &lt;call&gt;.
   */
  public static Timeline<? extends Number> createAgeTimeline(
      final Session session, final Date dateOfBirth) {

    /**
     * El límite artificial, de modo que la línea de tiempo tenga un número finito
     * de cambios de valor.
     */
    final int NUMBER_OF_BIRTHDAYS = 200;

    final Collection<Interval<Integer>> intervals =
        new ArrayList<Interval<Integer>>(NUMBER_OF_BIRTHDAYS + 2);

    /*
     * la edad antes de la fecha de nacimiento se seguirá registrando como 0 -
     * crear una aplicación de intervalo inicial desde el
     * "inicio del tiempo"
     */
    final Interval<Integer> initialInterval =
        new Interval<Integer>(null, 0);
    intervals.add(initialInterval);

    /*
     * Identificar cada cumpleaños hasta el límite. Tenga en cuenta que se considera
     * que la persona tiene una edad de 0 incluso antes de la fecha de nacimiento (consulte
     * más arriba); de modo que aquí la línea de tiempo fusionará el intervalo desde la fecha de
     * nacimiento hasta el primer cumpleaños se fusionará en un solo intervalo;
     no importa (es más claro mantener la lógica como
     * está).
```

```
     */

    for (int age = 0; age <= NUMBER_OF_BIRTHDAYS; age++) {

      // calcular la fecha de nacimiento
      final Calendar birthdayCalendar = dateOfBirth.getCalendar();

      /*
       * NB utilizar .roll en lugar de .add para obtener el
       * proceso correcto para años bisiestos
       */
      birthdayCalendar.roll(Calendar.YEAR, age);
      final Date birthdayDate = new Date(birthdayCalendar);

      /*
       * la edad se aplica desde el nacimiento hasta el siguiente cumpleaños
       */
      intervals.add(new Interval<Integer>(birthdayDate, age));
    }

    final Timeline<Integer> ageTimeline =
        new Timeline<Integer>(intervals);

    return ageTimeline;

  }
}
```

Nota: En general, los clientes de CER tienden a crear los datos de tiempo de línea fuera de las reglas.

En particular, el proceso de elegibilidad/titularidad de Cúram V6 contiene lógica para ayudarle a convertir las pruebas de Cúram en datos de línea de tiempo.

Consulte la guía `Inside Cúram Eligibility and Entitlement Using Cúram Express Rules` para obtener más detalles.

Construcción de líneas de tiempo en las reglas CER: Normalmente, los datos de línea de tiempo los crean los clientes de CER fuera de las reglas y se utilizan para llenar el valor de un atributo CER utilizando el mecanismo de especificación.

Sin embargo, CER también contiene algunas expresiones para crear líneas de tiempo directamente en las reglas CER:

- `Línea de tiempo`, conjuntamente con `Intervalo` y
- `existencetimeline`.

Línea de tiempo e intervalo

Una línea de tiempo se puede crear de manera nativa en las reglas CER creando primero explícitamente una lista de intervalos y, a continuación, utilizando esta lista para crear una línea de tiempo.

En la práctica, estas líneas de tiempo fijas tienden a ser útiles sólo como medida temporal mientras se elabora el conjunto de reglas. Esto es un ejemplo de usingTimeline y el intervalo para crear una línea de tiempo

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_Timeline"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
```

```
<Class name="CreateTimelines">

  <!-- Este ejemplo utiliza <initialvalue> para establecer el valor válido
       desde el inicio del tiempo. -->
  <Attribute name="aNumberTimeline">
    <type>
      <javaclass name="curam.creole.value.Timeline">
        <javaclass name="Number"/>
      </javaclass>
    </type>
    <derivation>
      <Timeline>
        <intervaltype>
          <javaclass name="Number"/>
        </intervaltype>
        <!-- Valor desde el inicio del tiempo -->
        <initialvalue>
          <Number value="0"/>
        </initialvalue>
        <!-- Los intervalos restantes -->
        <intervals>
          <fixedlist>
            <listof>
              <javaclass name="curam.creole.value.Interval">
                <javaclass name="Number"/>
              </javaclass>
            </listof>
            <members>
              <Interval>
                <intervaltype>
                  <javaclass name="Number"/>
                </intervaltype>
                <start>
                  <Date value="2001-01-01"/>
                </start>
                <value>
                  <Number value="10000"/>
                </value>
              </Interval>
              <Interval>
                <intervaltype>
                  <javaclass name="Number"/>
                </intervaltype>
                <start>
                  <Date value="2004-12-01"/>
                </start>
                <value>
                  <Number value="12000"/>
                </value>
              </Interval>

            </members>
          </fixedlist>

        </intervals>
      </Timeline>

    </derivation>
  </Attribute>


  <!-- Este ejemplo no utiliza <initialvalue>. -->
  <Attribute name="aStringTimeline">
    <type>
      <javaclass name="curam.creole.value.Timeline">
        <javaclass name="String"/>
      </javaclass>
```

```
      </type>
      <derivation>
        <Timeline>
          <intervaltype>
            <javaclass name="String"/>
          </intervaltype>

          <!-- La lista de intervalos debe incluir uno válido desde la
             fecha nula (inicio de tiempo), de lo contrario, se producirá un error
             en el tiempo de ejecución, si se evalúa esta expresión. -->
          <intervals>
            <fixedlist>
              <listof>
                <javaclass name="curam.creole.value.Interval">
                  <javaclass name="String"/>
                </javaclass>
              </listof>
              <members>
                <Interval>
                  <intervaltype>
                    <javaclass name="String"/>
                  </intervaltype>
                  <start>
                    <!-- "desde el inicio del tiempo" -->
                    <null/>
                  </start>
                  <value>
                    <String value="Start of time string"/>
                  </value>
                </Interval>
                <Interval>
                  <intervaltype>
                    <javaclass name="String"/>
                  </intervaltype>
                  <start>
                    <Date value="2001-01-01"/>
                  </start>
                  <value>
                    <String value="2001-only String"/>
                  </value>
                </Interval>
                <Interval>
                  <intervaltype>
                    <javaclass name="String"/>
                  </intervaltype>
                  <start>
                    <Date value="2002-01-01"/>
                  </start>
                  <value>
                    <String value="2002-onwards String"/>
                  </value>
                </Interval>
              </members>
            </fixedlist>
          </intervals>
        </Timeline>
      </derivation>
    </Attribute>
  </Class>
</RuleSet>
```

existencetimeline

Algunos objetos de negocio tienen fechas de inicio y finalización naturales, que juntas especifican un periodo durante el cual el objeto de negocio *existe*. Una o las

dos fechas de inicio y finalización pueden ser opcionales, en cuyo caso el periodo de existencia para el objeto de negocio es de finalización abierta.

Los ejemplos pueden incluir:

- un empleo, que inicia y posteriormente finaliza;
- un activo, que se compra y posteriormente se vende; y
- una persona que nace y posteriormente muere.

Las fechas de inicio y finalización para un objeto de negocio pueden utilizarse para dividir el tiempo en estos tres periodos (o menos si la fecha de inicio o la fecha de finalización está en blanco):

- **Periodo previo a existencia**

  periodo de tiempo antes de la fecha de inicio de negocio (si existe la fecha de inicio);
- **Periodo de existencia**

  periodo de tiempo desde la fecha de inicio de negocio hasta la fecha de finalización de negocio inclusive;
- **Periodo posterior a existencia**

  periodo de tiempo después de la fecha de finalización de negocio (si existe la fecha de finalización).

Normalmente puede ser útil asignar un valor diferente a cada uno de estos periodos para un objeto de negocio y crear una línea de tiempo a partir de estos valores. CER contiene una expresión `existencetimeline` para crear una línea de tiempo de valores de previos a la existencia/de existencia/posteriores a la existencia basándose en fechas de inicio y finalización opcionales.

Si la fecha de inicio no existe, no habrá ningún intervalo previo a la existencia en la línea de tiempo. Por ejemplo, si un activo no tiene una fecha de compra registrada, el valor efectivo se aplicará desde el inicio del tiempo, sin periodo de "valor cero".

Si la fecha de finalización no existe, no habrá ningún intervalo posterior a la existencia en la línea de tiempo. Por ejemplo, si un activo no tiene una fecha de venta y, a continuación, el valor del activo se mantiene hasta nuevo aviso (es decir, arbitrariamente en el futuro)

Consulte “Existence Timeline” en la página 129 para obtener detalles sobre cómo utilizar la línea de tiempo de existencia en el Editor CER.

Operación en líneas de tiempo

Las líneas de tiempo CER son útiles para almacenar datos que varían a lo largo del tiempo. CER soporta una característica de "Operación de línea de tiempo" que permite que las expresiones CER operen en elementos de datos de línea de tiempo para producir resultados de línea de tiempo.

Conservación de fechas de cambio: Todas las expresiones pueden operar en líneas de tiempo de una forma en la que las fechas de cambio para los valores de línea de tiempo de entrada se correlacionan de manera natural a través de las fechas de cambio para el valor de tiempo de línea resultante.

En los ejemplos anteriores se ha presentado el concepto de operación en las líneas de tiempo (línea de tiempo *se casó*, línea de tiempo *tiene un dependiente de menos de 16 años de edad*) para producir una línea de tiempo de salida (*es progenitor solo de un menor de edad*).

Más formalmente, para cualquier expresión CER que opera en uno o más valores, CER permite que la expresión opere también en una línea de tiempo de esos valores. En general, cualquier operación que se puede aplicar a tipos primitivos (por ejemplo fecha, número, serie, booleano, etc.) para obtener un resultado, se puede aplicar en su lugar a líneas de tiempo de esos tipos (por ejemplo Línea de tiempo<Fecha>, Línea de tiempo<Número>, Línea de tiempo<Serie>, Línea de tiempo<Booleano>) para obtener un resultado que sea un valor de Línea de tiempo.

CER contiene expresiones especiales denominadas "`timelineoperation`" en la página 239 y "`intervalvalue`" en la página 203 que hacen que las demás expresiones CER no "sepan" que están operando en Líneas de tiempo.

Por ejemplo, CER contiene una expresión "`sum`" en la página 234 para añadir una lista de números. Si una persona tiene varios ingresos, podemos sumar esos ingresos en un punto en el tiempo para obtener los ingresos totales de la persona en ese punto en el tiempo. Sin embargo, si lugar de ello tenemos *líneas de tiempo* de cómo esos importes de ingresos cambian con el tiempo, simplemente podemos utilizar de manera igual de fácil la expresión "`sum`" en la página 234 para obtener la forma en que los ingresos totales cambian lo largo del tiempo:

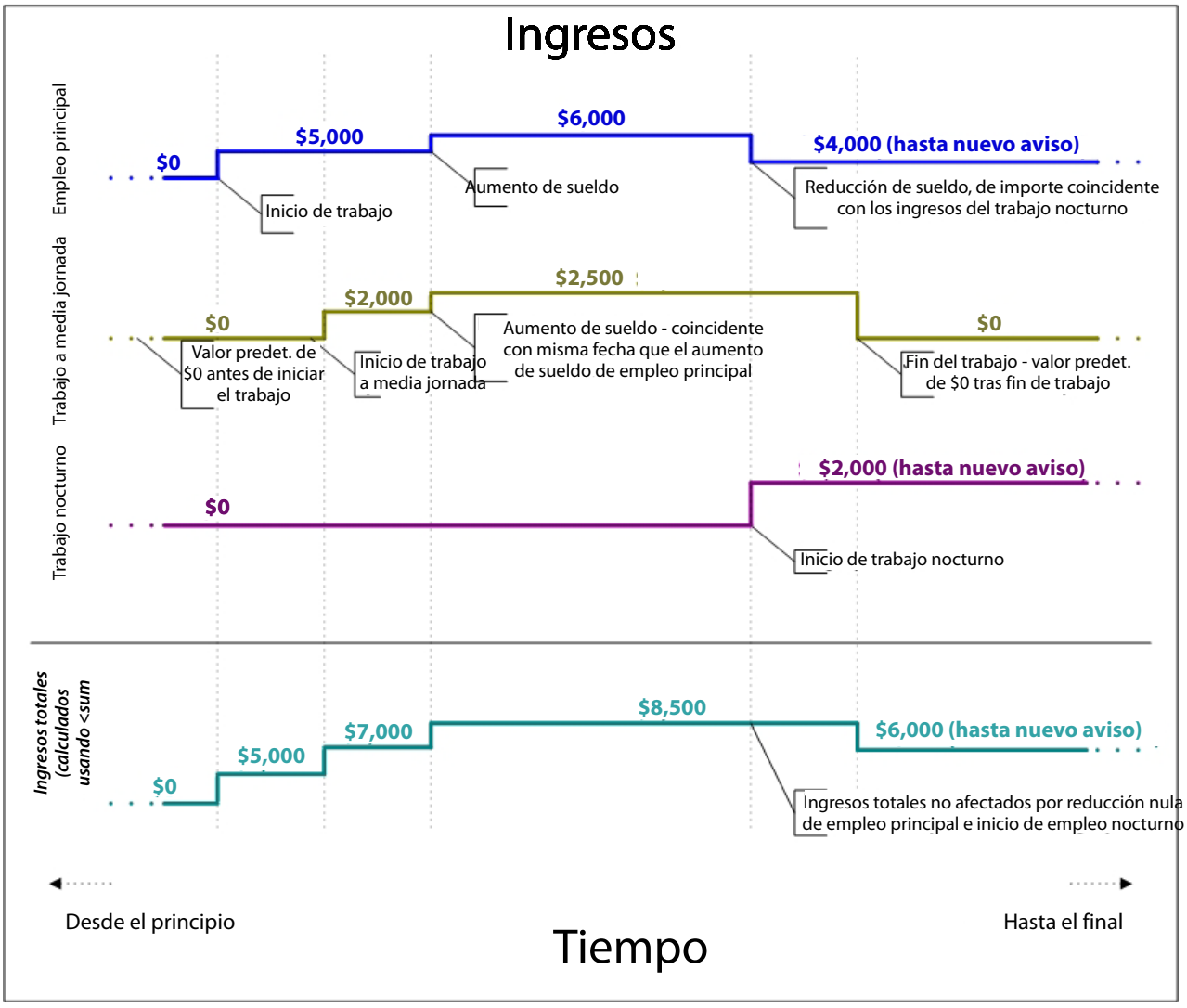

Figura 13. Una línea de tiempo de ingresos totales, calculados utilizando `sum`

Desplazamiento de fecha: En función de los requisitos de negocio, es posible que necesite crear una línea de tiempo basada en otra línea de tiempo, donde las fechas de cambio de valor en la línea de tiempo resultante sean diferentes de las de la línea de tiempo de entrada.

CER no incluye expresiones para el desplazamiento de fecha, porque los tipos de desplazamiento de fecha necesarios tienden a ser específicos del negocio. El enfoque recomendado es crear un método Java estático para crear la línea de tiempo necesaria e invocar el método estático de reglas mediante el uso de la expresión "`call`" en la página 175.

Importante: Al implementar un algoritmo de desplazamiento de fecha, asegúrese de que no existe ningún intento de crear una línea de tiempo con más de un valor en cualquier fecha determinada, ya que un intento de este tipo fallará en el tiempo de ejecución.

Las pruebas para el algoritmo deben incluir pruebas para casos límite, por ejemplo años bisiestos o meses que tienen un número de días diferente.

Ejemplo de adición de fecha

Tiene el siguiente requisito de negocio: una persona no puede solicitar un tipo de prestación en tres meses desde la recepción de dicha prestación.

Para implementar este requisito de negocio, ya tiene una línea de tiempo `isReceivingBenefitTimeline` que muestra los periodos de tiempo durante los cuales una persona está recibiendo la prestación.

Ahora necesita otra línea de tiempo `isDisallowedFromApplyingForBenefitTimeline` que muestre los periodos en que no es válido que dicha persona vuelva a solicitar la prestación. Esta línea de tiempo es una adición de fecha de 3 meses a las fechas de cambio de valor en `isReceivingBenefitTimeline`:

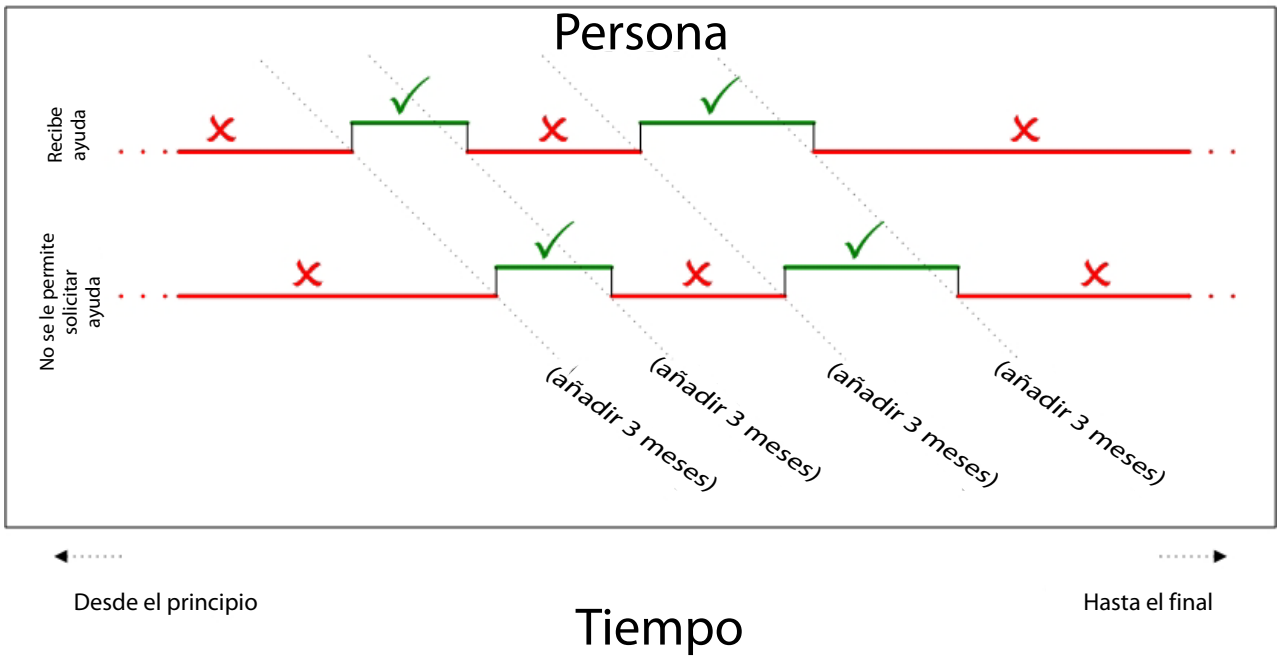

Figura 14. Requisito para una línea de tiempo de adición de fecha

A continuación se muestra una implementación de ejemplo de un método estático al que se puede llamar desde las reglas CER:

```
package curam.creole.example;

import java.util.Calendar;
import java.util.Collection;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import curam.creole.execution.session.Session;
import curam.creole.value.Interval;
import curam.creole.value.Timeline;
import curam.util.type.Date;

public class DateAdditionTimeline {

  /**
   * Crea una línea de tiempo basada en la línea de tiempo de entrada, con la fecha
   * desplazada en el número de meses especificados.
   * <p>
```

```
 * Tenga en cuenta que el parámetro de línea de tiempo puede ser de cualquier tipo.
 *
 * @param session
 *          la sesión CER
 * @param inputTimeline
 *          la línea de tiempo cuyas fechas se deben desplazar
 * @param monthsToAdd
 *          el número de meses a añadir a las fechas de cambio de
 *          línea de tiempo
 * @param <VALUE>
 *          el tipo de valor mantenido en las líneas de tiempo de entrada/salida
 * @return una línea de tiempo nueva con los valores de la línea
 *         de tiempo de entrada, desplazada por el número de meses especificados
 */
public static <VALUE> Timeline<VALUE> addMonthsTimeline(
    final Session session, final Timeline<VALUE> inputTimeline,
    final Number monthsToAdd) {

  /*
   * Normalmente CER pasará un número, que se debe convertir en
   * un entero
   */
  final int monthsToAddInteger = monthsToAdd.intValue();

  /*
   * Buscar los intervalos en la línea de tiempo de entrada
   */
  final List<? extends Interval<VALUE>> inputIntervals =
      inputTimeline.intervals();

  /*
   * Reunir los intervalos de salida. Observe que correlacionamos por fecha de inicio,
   * porque al añadir meses, es posible que varias
   * fechas de entrada diferentes se desplacen a la misma fecha de salida.
   *
   * Por ejemplo 3 meses después de estas fechas: 28-11-2002,
   * 29-11-2002, 30-11-2002, se calculan todas como 28-02-2003
   *
   * En esta situación, se utiliza el valor de la primera fecha de entrada
   * sólo - las fechas de entrada se procesan en orden ascendente
   */
  final Map<Date, Interval<VALUE>> outputIntervalsMap =
      new HashMap<Date, Interval<VALUE>>(inputIntervals.size());

  for (final Interval<VALUE> inputInterval : inputIntervals) {
    // obtener la fecha de inicio de intervalo
    final Date inputStartDate = inputInterval.startDate();

    /*
     * Añadir el número de meses - pero n meses después del inicio de
     * tiempo sigue siendo el inicio de tiempo
     */

    final Date outputStartDate;
    if (inputStartDate == null) {
      outputStartDate = null;
    } else {
      final Calendar startDateCalendar =
          inputStartDate.getCalendar();

      startDateCalendar.add(Calendar.MONTH, monthsToAddInteger);
      outputStartDate = new Date(startDateCalendar);

    }

    // comprobar que esta fecha de salida aún no se ha procesado
    if (!outputIntervalsMap.containsKey(outputStartDate)) {
```

```
        /*
         * el intervalo de salida utiliza el mismo valor que el intervalo de
         * entrada, pero con una fecha de inicio desplazada
         */

        final Interval<VALUE> outputInterval =
            new Interval<VALUE>(outputStartDate,
                inputInterval.value());
        outputIntervalsMap.put(outputStartDate, outputInterval);
      }
    }

    // crear una línea de tiempo desde intervalos de salida
    final Collection<Interval<VALUE>> outputIntervals =
        outputIntervalsMap.values();
    final Timeline<VALUE> outputTimeline =
        new Timeline<VALUE>(outputIntervals);
    return outputTimeline;

  }
}
```

Ejemplo de dispersión de fecha

Tiene este requisito empresarial: se debe gravar un coche por cada mes en el que el coche esté “en la carretera” durante uno o más días.

Nota: Si el coche se vuelve a poner en circulación de forma parcial después de un mes, el poseedor del coche debe asegurarse de que el impuesto se paga de forma retroactiva para todo el mes.

Para implementar este requisito de negocio, ya tiene una línea de tiempo `isOnRoadTimeline` que muestra los periodos de tiempo durante los cuales un automóvil está “circulando”.

Ahora necesita otra línea de tiempo `taxDueTimeline` que muestre los periodos en que se debe gravar al automóvil. Esta línea de tiempo es una dispersión de las fechas en `isOnRoadTimeline`:

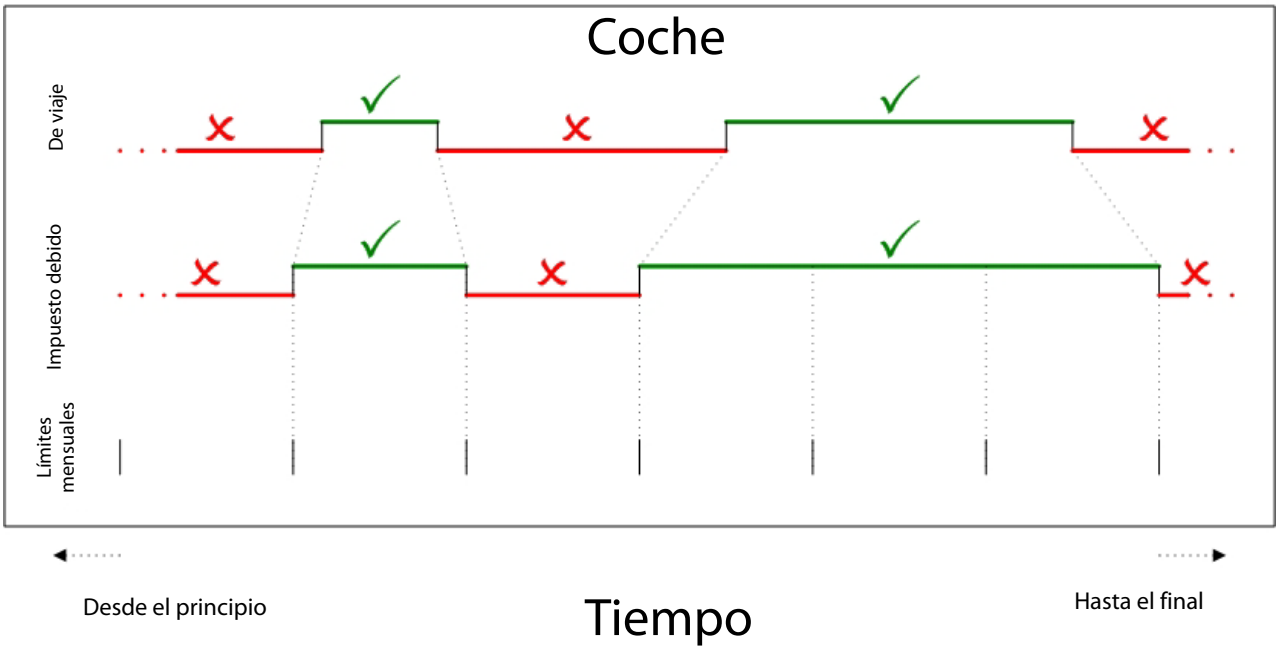

Figura 15. Requisito para una línea de tiempo de dispersión de fecha

A continuación se muestra una implementación de ejemplo de un método estático al que se puede llamar desde las reglas CER:

```
package curam.creole.example;

import java.util.Calendar;
import java.util.Collection;
import java.util.GregorianCalendar;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import curam.creole.execution.session.Session;
import curam.creole.value.Interval;
import curam.creole.value.Timeline;
import curam.util.type.Date;

public class DateSpreadingTimeline {

  /**
   * Crea una línea de tiempo para el periodo durante el cual un automóvil debe
   * gravarse.
   * <p>
   * El automóvil debe gravarse durante el mes entero en cualquier mes en el que
   * dicho automóvil esté circulando durante uno o más días durante ese
   * mes.
   */
  public static Timeline<Boolean> taxDue(final Session session,
      final Timeline<Boolean> isOnRoadTimeline) {

    /*
     * Buscar los intervalos en la línea de tiempo de entrada
     */
    final List<? extends Interval<Boolean>> isOnRoadIntervals =
        isOnRoadTimeline.intervals();

    /*
     * Reunir los intervalos de salida. Observe que correlacionamos por fecha de inicio;
     * un automóvil puede estar fuera de circulación durante un mes, lo que implicaría
```

```
     * que no sería necesario ningún impuesto al principio del mes siguiente, pero
     * vuelve a estar en circulación durante una parte del mes siguiente, en
     * cuyo caso se debe gravar a pesar de todo.
     *
     * Por ejemplo, el automóvil vuelve a estar circulando el 15-01-2001, de modo que el impuesto
     * es necesario (retrospectivamente) desde el 01-01-2001.
     *
     * El 24-01-2001, el automóvil vuelve estar fuera de circulación, de modo que es
     * posible que no necesite gravarse a partir del
     * 01-02-2001.
     *
     * Sin embargo, el 05-02-2001-02-05 el automóvil vuelve a circular y
     * por lo tanto necesita gravarse a partir del 01-02-2001. La
     * línea de tiempo resultante fusionará estos periodos para mostrar que el
     * automóvil necesita gravarse a partir del 01-01-2001 en adelante (cubriendo así
     * también desde el 01-02-2001).
     */
    final Map<Date, Interval<Boolean>> taxDueIntervalsMap =
        new HashMap<Date, Interval<Boolean>>(
            isOnRoadIntervals.size());

   for (final Interval<Boolean> isOnRoadInterval :
isOnRoadIntervals) {
     // obtener la fecha de inicio de intervalo
     final Date isOnRoadStartDate = isOnRoadInterval.startDate();

     if (isOnRoadStartDate == null) {
       // al principio de la hora, el automóvil se debe gravar si está
       // circulando
       taxDueIntervalsMap.put(null, new Interval<Boolean>(null,
           isOnRoadInterval.value()));
     } else if (isOnRoadInterval.value()) {
       /*
        * inicio de un periodo de circulación del automóvil - el automóvil
        * se debe gravar desde el inicio del mes que contiene el
        * inicio de este periodo
        */

       final Calendar carOnRoadStartCalendar =
           isOnRoadStartDate.getCalendar();
       final Calendar startOfMonthCalendar =
           new GregorianCalendar(
               carOnRoadStartCalendar.get(Calendar.YEAR),
               carOnRoadStartCalendar.get(Calendar.MONTH), 1);
       final Date startOfMonthDate =
           new Date(startOfMonthCalendar);

       /*
        * Añadir al mapa de periodos de vencimiento de impuestos - tenga en cuenta que esto
        * extraerá del mapa cualquier intervalo de "impuesto no vencido"
        * añadido de forma especulativa si el automóvil ha estado fuera de circulación durante
        * el mes anterior
        */
       taxDueIntervalsMap.put(startOfMonthDate,
           new Interval<Boolean>(startOfMonthDate, true));
     } else {
       /*
        * Inicio de un periodo en el que el automóvil ha estado fuera de circulación -
        * especular que desde el inicio del siguiente mes, el automóvil puede
        * no requerir impuestos. Esta especulación se mantendrá a menos que
        * posteriormente se encuentre que el automóvil está de nuevo circulando
        * el siguiente mes, en cuyo caso esta especulación se
        * descartará (es decir, se extraerá del mapa).
        */

       final Calendar carOffRoadStartCalendar =
           isOnRoadStartDate.getCalendar();
```

```
        final Calendar startOfNextMonthCalendar =
            new GregorianCalendar(
                carOffRoadStartCalendar.get(Calendar.YEAR),
                carOffRoadStartCalendar.get(Calendar.MONTH), 1);
        startOfNextMonthCalendar.add(Calendar.MONTH, 1);

        final Date startOfNextMonthDate =
            new Date(startOfNextMonthCalendar);

        /*
         * Añadir al mapa de periodos de vencimiento de impuestos - tenga en cuenta que esto
         * extraerá del mapa cualquier intervalo de "impuesto no vencido"
         * añadido de forma especulativa si el automóvil ha estado fuera de circulación durante
         * el mes anterior
         */
        taxDueIntervalsMap.put(startOfNextMonthDate,
            new Interval<Boolean>(startOfNextMonthDate, false));
      }

    }

    // crear una línea de tiempo desde los intervalos de vencimiento de impuestos
    final Collection<Interval<Boolean>> taxDueIntervals =
        taxDueIntervalsMap.values();
    final Timeline<Boolean> taxDueTimeline =
        new Timeline<Boolean>(taxDueIntervals);
    return taxDueTimeline;

  }
}
```

Prueba de salidas de línea de tiempo

Un atributo de regla que devuelve una línea de tiempo de valores deben probarse en las pruebas JUnit de forma similar a las pruebas para valores primitivos (no de línea de tiempo).

Para simplificar las pruebas, no necesita probar que "`timelineoperation`" en la página 239 acumula correctamente las fechas de cambio (a menos que lo desee). Para que las pruebas sean simples, generalmente puede utilizar líneas de tiempo de entrada que tengan un valor constante para siempre.

Por ejemplo, digamos que tiene un atributo de regla que calcula (en forma de línea de tiempo) el total de una lista de números:

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_NumberSumTimeline"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">

  <Class name="Totalizer">

    <!-- Las líneas de tiempo a sumar -->
    <Attribute name="inputNumberTimelines">
      <type>
        <javaclass name="List">
          <javaclass name="curam.creole.value.Timeline">
            <javaclass name="Number"/>
          </javaclass>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>
```

```
    <!-- El total resultante -->
    <Attribute name="totalTimeline">
      <type>
        <javaclass name="curam.creole.value.Timeline">
          <javaclass name="Number"/>
        </javaclass>
      </type>
      <derivation>
        <timelineoperation>
          <sum>
            <dynamiclist>
              <list>
                <reference attribute="inputNumberTimelines"/>
              </list>
              <listitemexpression>
                <intervalvalue>
                  <current/>
                </intervalvalue>
              </listitemexpression>
            </dynamiclist>

          </sum>
        </timelineoperation>
      </derivation>
    </Attribute>

  </Class>
</RuleSet>
```

Puede escribir una prueba simple que utiliza líneas de tiempo de entrada que tienen siempre un valor constante:

```
package curam.creole.example;

import java.util.Arrays;

import junit.framework.TestCase;
import curam.creole.calculator.CREOLETestHelper;
import curam.creole.execution.session.RecalculationsProhibited;
import curam.creole.execution.session.Session;
import curam.creole.execution.session.Session_Factory;
import
 curam.creole.execution.session.StronglyTypedRuleObjectFactory;
import
 curam.creole.ruleclass.Example_NumberSumTimeline.impl.Totalizer;
import
 curam.creole.ruleclass.Example_NumberSumTimeline.impl.Totalizer_Factory;
import curam.creole.storage.inmemory.InMemoryDataStorage;
import curam.creole.value.Timeline;

public class TestForeverValuedTimelines extends TestCase {

  public void testNumberSumTimeline() {

    final Session session =
        Session_Factory.getFactory().newInstance(
            new RecalculationsProhibited(),
            new InMemoryDataStorage(
                new StronglyTypedRuleObjectFactory()));

    final Totalizer totalizer =
        Totalizer_Factory.getFactory().newInstance(session);

    // utilizar valores de entrada que no varían a lo largo del tiempo

    final Timeline<Number> inputTimeline1 =
```

```
        new Timeline<Number>(1);
    final Timeline<Number> inputTimeline2 =
        new Timeline<Number>(2);
    final Timeline<Number> inputTimeline3 =
        new Timeline<Number>(3);

    totalizer.inputNumberTimelines().specifyValue(
        Arrays.asList(inputTimeline1, inputTimeline2,
            inputTimeline3));

    // comprobar que la línea de tiempo resultante es siempre 6
    CREOLETestHelper.assertEquals(new Timeline<Number>(6),
        totalizer.totalTimeline().getValue());

  }
}
```

Consejo: La clase `Timeline` tiene un constructor preparado para crear una línea de tiempo con un valor constante para siempre.

En algunas situaciones, por ejemplo cuando ha creado su propio algoritmo de desplazamiento de fecha o si realmente necesita probar que las fechas de cambio para las líneas de tiempo de entrada están correctamente reflejadas en las líneas de tiempo resultantes, hay diferentes enfoques que puede adoptar según sus necesidades:

- **Comprobación estricta**

  Compruebe que la línea de tiempo resultante es exactamente igual a un valor de línea de tiempo esperado. (La semántica de igualdad de la clase `Timeline` se comporta como se espera - dos líneas de tiempo son iguales si contienen exactamente la misma colección de intervalos, es decir los valores de las dos líneas de tiempo sin idénticos para cada fecha posible).
- **Comprobación flexible**

  Compruebe que la línea de tiempo resultante tiene el valor que espera en fechas determinadas.

Este ejemplo muestra las pruebas estrictas de una línea de tiempo resultante.

```
package curam.creole.example;

import java.util.Arrays;

import junit.framework.TestCase;
import curam.creole.calculator.CREOLETestHelper;
import curam.creole.execution.session.RecalculationsProhibited;
import curam.creole.execution.session.Session;
import curam.creole.execution.session.Session_Factory;
import
 curam.creole.execution.session.StronglyTypedRuleObjectFactory;
import
 curam.creole.ruleclass.Example_NumberSumTimeline.impl.Totalizer;
import
 curam.creole.ruleclass.Example_NumberSumTimeline.impl.Totalizer_Factory;
import curam.creole.storage.inmemory.InMemoryDataStorage;
import curam.creole.value.Interval;
import curam.creole.value.Timeline;
import curam.util.type.Date;

public class TestStrictTimelineChecking extends TestCase {

  public void testNumberSumTimeline() {

    final Session session =
        Session_Factory.getFactory().newInstance(
```

```
            new RecalculationsProhibited(),
            new InMemoryDataStorage(
                new StronglyTypedRuleObjectFactory()));

    final Totalizer totalizer =
        Totalizer_Factory.getFactory().newInstance(session);

    // utilizar valores de entrada que varían a lo largo del tiempo

    final Timeline<Number> inputTimeline1 =
        new Timeline<Number>(Arrays.asList(
          new Interval<Number>(null, 1),
          new Interval<Number>(Date.fromISO8601("20010101"), 1.1)
        ));

    final Timeline<Number> inputTimeline2 =
        new Timeline<Number>(Arrays.asList(
          new Interval<Number>(null, 2),
          new Interval<Number>(Date.fromISO8601("20020101"), 2.2)
        ));

    final Timeline<Number> inputTimeline3 =
        new Timeline<Number>(Arrays.asList(
          new Interval<Number>(null, 3),
          new Interval<Number>(Date.fromISO8601("20030101"), 3.3)
         ));

    totalizer.inputNumberTimelines().specifyValue(
        Arrays.asList(inputTimeline1, inputTimeline2,
            inputTimeline3));

    // comprobar estrictamente el valor exacto de la línea de tiempo resultante
    CREOLETestHelper.assertEquals(

    new Timeline<Number>(Arrays.asList(
      new Interval<Number>(null, 6),
      new Interval<Number>(Date.fromISO8601("20010101"), 6.1),
      new Interval<Number>(Date.fromISO8601("20020101"), 6.3),
      new Interval<Number>(Date.fromISO8601("20030101"), 6.6)
    )),

    totalizer.totalTimeline().getValue());

  }
}
```

El ejemplo muestra pruebas más flexibles de una línea de tiempo resultante.

```
package curam.creole.example;

import java.util.Arrays;

import junit.framework.TestCase;
import curam.creole.calculator.CREOLETestHelper;
import curam.creole.execution.session.RecalculationsProhibited;
import curam.creole.execution.session.Session;
import curam.creole.execution.session.Session_Factory;
import
 curam.creole.execution.session.StronglyTypedRuleObjectFactory;
import
 curam.creole.ruleclass.Example_NumberSumTimeline.impl.Totalizer;
import
 curam.creole.ruleclass.Example_NumberSumTimeline.impl.Totalizer_Factory;
import curam.creole.storage.inmemory.InMemoryDataStorage;
import curam.creole.value.Interval;
import curam.creole.value.Timeline;
import curam.util.type.Date;
```

```
public class TestLaxTimelineChecking extends TestCase {

  public void testNumberSumTimeline() {

    final Session session =
        Session_Factory.getFactory().newInstance(
            new RecalculationsProhibited(),
            new InMemoryDataStorage(
                new StronglyTypedRuleObjectFactory()));

    final Totalizer totalizer =
        Totalizer_Factory.getFactory().newInstance(session);

    // utilizar valores de entrada que varían a lo largo del tiempo

    final Timeline<Number> inputTimeline1 =
        new Timeline<Number>(Arrays.asList(
          new Interval<Number>(null, 1),
          new Interval<Number>(Date.fromISO8601("20010101"), 1.1)
        ));

    final Timeline<Number> inputTimeline2 =
        new Timeline<Number>(Arrays.asList(
          new Interval<Number>(null, 2),
          new Interval<Number>(Date.fromISO8601("20020101"), 2.2)
        ));

    final Timeline<Number> inputTimeline3 =
        new Timeline<Number>(Arrays.asList(
          new Interval<Number>(null, 3),
          new Interval<Number>(Date.fromISO8601("20030101"), 3.3)
        ));

    totalizer.inputNumberTimelines().specifyValue(
        Arrays.asList(inputTimeline1, inputTimeline2,
          inputTimeline3));

    /*
     * No comprobar estrictamente que la línea de tiempo resultante es exactamente
     * como se esperaba; en lugar de ello comprobar el valor de la línea de tiempo resultante
     * en fechas determinadas.
     *
     * Es posible que la línea de tiempo tenga valores incorrectos en
     * otras fechas, pero según el propósito de la prueba,
     * es aconsejable que cambie la rigurosidad por una mejor legibilidad.
     */

    final Timeline<? extends Number> resultantTimeline =
        totalizer.totalTimeline().getValue();
    CREOLETestHelper.assertEquals(6.1,
        resultantTimeline.valueOn(Date.fromISO8601("20010101")));
    CREOLETestHelper.assertEquals(6.6,
        resultantTimeline.valueOn(Date.fromISO8601("20130101")));
  }
}
```

Propiedades de línea de tiempo

Cada línea de tiempo de CER tiene algunas propiedades importantes que debe conocer antes de trabajar con líneas de tiempo en los conjuntos de reglas CER, conjuntos de reglas y cualquier código que sea un cliente de CER:

- cada línea de tiempo CER es inmutable (como todos los tipos de datos utilizados en CER);
- cada referencia a la línea de tiempo se parametriza con el tipo de valor mantenido en la línea de tiempo, que pueden ser primitivas como String, Date,

Number, Boolean etc. o un tipo complejo arbitrario, como una clase de regla u otro tipo parametrizado como List. En cuanto a otros tipos parametrizados en CER, el propio parámetro debe ser un objeto inmutable;

- cada línea de tiempo de CER se extiende infinitamente lejos en el pasado y también en el futuro. En otras palabras, cada línea de tiempo de CER tiene un valor en *cualquier* fecha, sin importar lo lejos que pueda estar esa fecha en el pasado o el futuro;

  Nota: Cada línea de tiempo cubre una cantidad infinita de tiempo, pero sólo puede contener un número finito de fechas en las que cambia su valor.
- cuando se crea una línea de tiempo de CER, esta se divide en una colección de *intervalos*, con cada uno de los cuales que contiene un valor constante para un periodo de tiempo dentro de la línea de tiempo. Los intervalos contiguos *siempre* tienen valores diferentes, de lo contrario, se fusionarían en un único intervalo. Cada línea de tiempo de CER se extiende infinitamente lejos en el pasado y también en el futuro;

  Nota: La semántica de Java `Object.equals(...)` `ha detectado valores iguales o diferentes.`. Todos los tipos que se han utilizado como tipo parametrizado en la línea de tiempo deben tener implementaciones sensibles de `Object.equals(...)` y `Object.hashCode()`.

Hay varias consecuencias de estas propiedades:

- no es posible tener un "hueco" en medio de una línea de tiempo: todos los intervalos de una línea de tiempo son contiguos;
- no es posible tener una línea de tiempo que se inicia en una fecha determinada; en determinadas circunstancias será necesario elegir un valor predeterminado sensible, por ejemplo si tiene una línea de tiempo<Número> para los ingresos que proceden de un empleo, esos ingresos deben ser 0 en todas las fechas antes de la fecha de inicio de empleo;
- no es posible tener una línea de tiempo que finalice en una fecha determinada: el último valor de la línea de tiempo se aplica "hasta nuevo aviso", es decir de forma arbitraria en el futuro; en determinadas circunstancias se deberá elegir un valor predeterminado razonable, por ejemplo si tiene una línea de tiempo<Número> para los ingresos procedentes de un empleo, si hay una fecha final conocida para ese empleo, los ingresos deben ser 0 en todas las fechas posteriores a la finalización del empleo; de lo contrario, si no hay ninguna fecha final conocida para ese empleo, se deben aplicar los ingresos más recientes hasta nuevo aviso;
- fallará cualquier intento de crear una línea de tiempo que no tenga un valor para cualquier fecha. En particular, cada línea de tiempo debe tener un valor que se aplique desde el *inicio del tiempo*, representado por una fecha de inicio de `null`;
- cada línea de tiempo puede contener un número finito de cambios de valor. Esto presenta una limitación para las líneas de tiempo que representan valores que cambian un número arbitrario de veces. Por ejemplo, puede tener una línea de tiempo<Número> para representar la edad de una persona, con el valor 0 hasta el primer cumpleaños de la persona, el valor 1 hasta el segundo cumpleaños de la persona y así sucesivamente. Para las Personas que siguen vivas no es posible predecir cuántos cumpleaños celebrarán y por lo tanto se debe imponer un límite práctico (por ejemplo, 200). En la práctica, esta limitación no debe presentar ninguna dificultad.

Ejemplo de intervalos de línea de tiempo: En el ejemplo de circunstancias de Juan, María y Jaime, vimos que María no era un progenitor solo de un menor de edad antes de casarse con Juan y que cuando se casó con Juan seguía no siendo un progenitor solo de un menor de edad, pero por razones diferentes.

Cuando se está calculando el valor `isLoneParentOfMinorTimeline` de María, las líneas de tiempo de entrada que se utilizan son la `isMarriedTimeline` de María y su `hasMinorDependentsTimeline`.

CER identifica cada fecha en la que cambian las líneas de tiempo de entrada; para cada una de esas fechas calcula el valor resultante (en esa fecha) para determinar si María es un progenitor solo de un menor de edad en esa fecha, como se indica a continuación:

- Fechas en las que `isMarriedTimeline` de María cambia:
  - 1 de enero de 2001 y
  - 1 de mayo de 2004.
- Fechas en las que `hasMinorDependentsTimeline` de María cambia:
  - 1 de enero de 2001 y
  - 1 de junio de 2006.
- Por consiguiente, las fechas en las que han cambiado una o más entradas son:
  - 1 de enero de 2001 (ambas líneas de tiempo de entrada de María cambian en esta fecha) y
  - 1 de mayo de 2004 (sólo `isMarriedTimeline` de María cambia en esta fecha) y
  - 1 de junio de 2006 (sólo `hasMinorDependentsTimeline` de María cambia en esta fecha).

Por lo tanto, para cada una de estas fechas, calcule el valor necesario para `isLoneParentOfMinorTimeline`, utilizando la lógica de tabla de decisión lógica/booleana de primitiva:

Tabla 2. Cálculo de valores de intervalo para el valor `isLoneParentOfMinorTimeline` de María

Fecha en la que una o más líneas de tiempo de entrada cambian de valor	**Valor de `isMarriedTimeline` en esta fecha**	**Valor de `hasMinorDependentsTimeline` en esta fecha**	**Valor necesario de `isLoneParentOfMinorTimeline` en esta fecha**
inicio de tiempo (esta fecha siempre se incluye)	FALSE	FALSE	FALSE
1 de enero de 2001	TRUE	TRUE	FALSE
1 de mayo de 2004	FALSE	TRUE	TRUE
1 de junio de 2006	FALSE	FALSE	FALSE

Por último, se construye una línea de tiempo con los valores necesarios para `isLoneParentOfMinorTimeline`; en este punto de la construcción de la línea de tiempo reconoce que el valor de inicio del tiempo (FALSE) y de 1 de enero de 2001 (FALSE) son idénticos y estos intervalos se fusionan en un único intervalo que se extiende desde el inicio del tiempo hasta (pero sin incluir) el 1 de mayo de 2004 (cuando el valor se convierte en TRUE).

Nota: La línea de tiempo resultante sólo tiene cambios de valor el 1 de mayo de 2004 y 1 de junio de 2006.

La línea de tiempo *no* contiene adrede ningún registro de que se utilizara el 1 de enero de 2001 durante la construcción, porque el valor de la línea de tiempo no cambió en esa fecha: esa fecha no tiene ninguna relevancia en absoluto para la línea de tiempo resultante.

Desencadenamiento de recálculo cuando cambian datos

La posibilidad de que CER efectúe recálculos directamente la reemplaza ahora el Gestor de dependencias (consulte "Gestor de dependencias").

Se recomienda utilizar el Gestor de dependencias en vez de las estrategias de recálculo de CER.

Gestor de dependencias

La aplicación incluye un Gestor de dependencias que es responsable de almacenar y gestionar dependencias entre elementos de datos de entrada ("precedentes") y elementos de datos de salida ("dependientes").

CER y sus clientes (como el Motor de elegibilidad y titularidad y el Asesor) se integran estrechamente con el Gestor de dependencias para soportar el recálculo de resultados CER siempre que cambian las entradas que se han utilizado en los cálculos CER.

En este capítulo se proporciona una visión general del Gestor de dependencias como se indica a continuación:

- los conceptos subyacentes del Gestor de dependencias y la terminología utilizada para describirlos;
- las funciones que el gestor de dependencia realiza;
- el proceso por lotes incluido con el Gestor de dependencias;
- cómo CER se integra con el Gestor de dependencias y
- conformidad para el Gestor de dependencias.

Conceptos del Gestor de dependencias

Siempre que el valor de un elemento de datos se deriva de los valores de uno o varios otros elementos de datos, podemos decir que el valor derivado *depende* de los valores utilizados para derivarlo. Si posteriormente cambian uno o varios de los valores de los que depende, se debe recalcular el elemento de datos derivado para obtener el nuevo valor.

El gestor de dependencias utiliza los términos siguientes para encapsular estos conceptos:

- **Dependiente**

  Elemento de datos derivado cuyo valor se calcula a partir de otros elementos de datos (precedentes).
- **Precedente**

  Elemento de datos cuyo valor puede utilizarse para calcular elementos de datos derivados (dependientes).
- **Dependencia**

Registro del hecho de que el valor de un determinado dependiente depende del valor de un precedente en particular.

- **Elemento de cambio de precedente**

  Registro del hecho de que el valor de un precedente en particular se ha modificado de alguna manera.
- **Conjunto de cambios de precedente**

  Conjunto de elementos de cambios de precedente, agrupados conjuntamente para el proceso; se utiliza para identificar dependientes potencialmente afectados que requieren recálculo.
- **Recálculo de dependiente**

  Recálculo de un dependiente que queda potencialmente afectado por uno o varios de los cambios en los precedentes de un conjunto de cambios de precedente.

Estos conceptos se describen mejor mediante un ejemplo.

Digamos que la titularidad de un demandante para beneficiarse se calcula a partir de datos tales como:

- detalles personales del demandante;
- pruebas reunidas para el caso del demandante y
- tarifas de prestación y umbrales de ingresos.

Juan, un demandante, tiene dos casos (123 y 124) y María, otro demandante, tiene un caso (125). También hay casos y detalles personales para otros demandantes y también algunas tarifas de pensión que se utilizan para otros cálculos.

En este ejemplo, la titularidad calculada para cada caso es un *dependiente* y los detalles personales, pruebas y tarifas/umbrales son *precedentes*.

Podemos trazar una *matriz dispersa* que muestre las dependencias entre los dependientes y los precedentes (una "X" indica la presencia de una dependencia):

Tabla 3. Ejemplo de matriz de dependencias

Precedente	**Titularidad del caso 123**	**Titularidad del caso 124**	**Titularidad del caso 125**	**Titularidad del caso 126**
Detalles personales de Juan	X	X		
Detalles personales de María			X	
Detalles personales de Fran				
Pruebas del caso 123	X			
Pruebas del caso 124		X		
Pruebas del caso 125			X	
Pruebas del caso 126				X

Tabla 3. Ejemplo de matriz de dependencias (continuación)

Precedente	Titularidad del caso 123	Titularidad del caso 124	Titularidad del caso 125	Titularidad del caso 126
Tarifas de prestación	X	X	X	X
Umbrales de ingresos	X	X	X	X
Tarifas de pensión				

Por lo tanto, para examinar algunos ejemplos de la matriz de dependencias:

- La titularidad para el caso 123 depende de los detalles personales de Juan, no de los de María;
- Los detalles personales de Juan se utilizan en el cálculo de los dos casos (123 y 124) y
- Todos los casos utilizan las tarifas de prestación y los umbrales de ingresos, pero no las tarifas de pensión.
- La titularidad de un caso no depende de los detalles personales de Fran.

Tenga en cuenta que la matriz se puede leer:

- por columna, para comprender todos los precedentes de los que depende un determinado dependiente: estos son el conjunto de dependencias que deben mantenerse cada vez que se calcula un dependiente y/o
- por fila, para comprender todos los dependientes que dependen de un precedente en particular: estos son el conjunto de dependientes que se debe recalcular cada vez que cambia el valor del precedente.

A medida que crece el número de precedentes y dependientes en el sistema, la matriz de dependencias aumenta de tamaño. Dado que la matriz sólo se ha llenado escasamente (es decir, cada dependiente depende sólo de una pequeña fracción de los precedentes disponible), los datos de la matriz se almacenan sólo para las dependencias que están presentes, como se indica a continuación:

Tabla 4. Ejemplo de almacenamiento de dependencias

Dependiente		Precedente
Titularidad del caso 123	depende de	Detalles personales de Juan
Titularidad del caso 123	depende de	Pruebas del caso 123
Titularidad del caso 123	depende de	Tarifas de prestación
Titularidad del caso 123	depende de	Umbrales de ingresos
Titularidad del caso 124	depende de	Detalles personales de Juan
Titularidad del caso 124	depende de	Pruebas del caso 124
Titularidad del caso 124	depende de	Tarifas de prestación
Titularidad del caso 124	depende de	Umbrales de ingresos
Titularidad del caso 125	depende de	Detalles personales de María
Titularidad del caso 125	depende de	Pruebas del caso 125
Titularidad del caso 125	depende de	Tarifas de prestación
Titularidad del caso 125	depende de	Umbrales de ingresos
Titularidad del caso 126	depende de	Pruebas del caso 126

Tabla 4. Ejemplo de almacenamiento de dependencias (continuación)

Dependiente		**Precedente**
Titularidad del caso 126	depende de	Tarifas de prestación
Titularidad del caso 126	depende de	Umbrales de ingresos

(Tenga en cuenta que la tabla anterior se muestra ordenada por dependiente, lo que hace que sea más fácil ver el conjunto de dependencias para cada dependiente, pero también se podría mostrar ordenada por precedente, lo que haría más fácil ver los dependientes potencialmente afectados por un cambio en el valor que dicho precedente.)

Supongamos que los detalles personales de Juan cambian. Puesto que las dependencias están registradas en los detalles personales de Juan, el Gestor de dependencia es capaz de identificar que los casos 123 y 124 requieren recálculo. Cuando se recalculan los casos, los valores de titularidad cambian (debido al cambio en los detalles personales de Juan); pero tenga en cuenta que en situaciones típicas, las propias dependencias no cambian: antes del recálculo, el caso 123 dependía de los detalles personales de Juan, de las pruebas almacenadas para el caso, de las tarifas de prestación y de los umbrales de ingresos y, después del recálculo lo mismo es cierto.

Es posible que cambie más de un valor de precedente de forma simultánea, por ejemplo, si la agencia opta por modificar las tarifas de prestación y los umbrales de ingresos, se deben recalcular todos los casos. Ingenuamente, cada caso se identificaría dos veces (una por el cambio en las tarifas de prestación y otra de nuevo por el cambio en los umbrales de ingresos); sin embargo, el Gestor de dependencias soporta la agrupación de estos dos cambios de precedentes en un conjunto de cambios de precedente. Cuando el Gestor de dependencias procesa el conjunto de cambios de precedente filtra automáticamente los dependientes duplicados identificados para realizar el trabajo mínimo necesario para recalcular los dependientes.

Funciones del Gestor de dependencias

El Gestor de dependencias realiza estas funciones principales:

- almacena registros de dependencias identificados por un cliente, por ejemplo por el motor de elegibilidad y titularidad al calcular una determinación de evaluación inicial;
- captura cambios en los valores de precedentes que potencialmente afectan a los valores de dependientes;
- identifica los dependientes que se ven potencialmente afectados por los elementos de un conjunto de cambios de precedente y
- controla el recálculo de estos dependientes identificados.

Estas funciones se describen más detalladamente en las secciones siguientes.

Almacenamiento de registros de dependencias

El gestor de dependencias es responsable de la creación de nuevos registros de dependencia en la base de datos y de la eliminación de registros de dependencia existentes que ya no son necesarios.

Nota: Cada registro de dependencia no contiene información modificable y el gestor de dependencias nunca *modifica* ningún registro de dependencia existente, sólo crea registros nuevos o elimina registros existentes.

Cada vez que un cliente del Gestor de dependencias calcula el valor de un dependiente, el cliente es responsable de identificar los precedentes utilizados en ese cálculo y de pasar ese dependiente y su conjunto de precedentes al Gestor de dependencias. El Gestor de dependencias utiliza ese dependiente para recuperar su conjunto existente de dependencias almacenadas (si hay alguno) desde la base de datos y crea o elimina registros de dependencia en línea con el nuevo conjunto de precedentes identificados por el cliente.

Normalmente, la primera vez que se invoca el Gestor de dependencias para un dependiente, el Gestor de dependencias creará varias filas nuevas en la base de datos para almacenar las dependencias de los precedentes identificados.

Sin embargo, en las invocaciones posteriores del Gestor de dependencias para el mismo dependiente, es muy común que el Gestor de dependencias encuentre que el nuevo conjunto de dependencias necesarias pasadas coincide exactamente con las que ya están almacenadas en la base de datos y, así, bajo estas circunstancias el Gestor de dependencias no debe realizar ninguna grabación de base de datos. Ocasionalmente el Gestor de dependencias encontrará que se necesita un número pequeño de filas de dependencia nuevas y/o que un pequeño número de filas de dependencia existentes son ahora extrañas y se deben eliminar; y, bajo estas circunstancias, el Gestor de dependencias realiza un pequeño número de grabaciones de base de datos para actualizar las filas almacenadas con las dependencias necesarias, dejando el grueso de los registros de dependencia sin cambios para el dependiente.

Los clientes del Gestor de dependencias pueden identificar qué registros de dependencias ya no son necesarios para un dependiente y pueden indicar al Gestor de dependencias que elimine todos los registros de dependencia de ese dependiente.

Ejemplo: Cuando se realiza por primera vez una titularidad de un caso, el Gestor de dependencias almacena los registros de dependencias nuevos para mostrar que la titularidad de caso depende de los detalles personales del demandante, las pruebas grabadas para el caso, las tarifas, etc.

Si posteriormente se vuelve a calcular el caso (digamos automáticamente por el gestor de dependencias en respuesta a un cambio en los detalles personales o manualmente solicitado por un usuario), después de recálculo el Gestor de dependencias compara las dependencias identificadas durante el cálculo con las que ya están en la base de datos y no se encuentra ninguna diferencia.

Si se añade un nuevo miembro de la unidad familiar al caso, cuando se vuelva a calcular la titularidad se identificará una nueva dependencia; es decir que la titularidad del caso ahora también depende de los detalles personales del nuevo miembro de la unidad familiar, además de las dependencias existentes ya almacenadas para el caso. El Gestor de dependencias crea un nuevo registro de dependencias en la base de datos para almacenar la dependencia adicional.

Si posteriormente se elimina el nuevo miembro de la unidad familiar, cuando se vuelva a calcular la titularidad no habrá ninguna dependencia de los detalles personales del miembro de la unidad familiar eliminado ahora. El Gestor de dependencias identifica que la dependencia almacenada en los detalles personales del miembro de la unidad familiar es ahora extraño y lo elimina de la base de datos, dejando los demás registros de dependencia (en los detalles personales del demandante, las pruebas grabadas para el caso, las tarifas, etc.) aún intactos.

Cuando finalmente se cierra el caso, dejará de necesitar soporte para volver a realizar cálculos, así pues, los registros de dependencias dejan de ser necesarios.

Nota: Se da por supuesto una estrategia de reevaluación de "No reevaluar casos cerrados". Consulte la guía *Inside Cúram Eligibility and Entitlement Using Cúram Express Rules*.
Para un buen mantenimiento, se invoca el Gestor de dependencias para eliminar todos los registros de dependencias para la titularidad del caso. Si posteriormente se reabre el caso, se puede volver a calcular la titularidad y el Gestor de dependencias vuelve a crear todos los registros de dependencias necesarios.

Sin conocimiento de los dependientes o precedentes: El Gestor de dependencias *no* mantiene intencionalmente ningún registro de todos los dependientes o precedentes conocidos en el sistema, para evitar que:

- se dupliquen los datos mantenidos en otros lugares del sistema y
- se convierta en un cuello de botella durante el proceso, produciendo potencialmente problemas de concurrencia cuando el sistema tiene conocimiento de que se utilizan datos nuevos como precedentes en los cálculos.

En lugar de ello, el Gestor de dependencias sólo registra información sobre las dependencias. Cada dependencia es solamente un enlace entre un dependiente concreto y un precedente concreto. Si un dependiente concreto no tiene precedentes, o viceversa, simplemente no se almacenarán registros de dependencia para el mismo.

El Gestor de dependencias no "conoce" la información de dependientes y precedentes almacenada en un registro de dependencia; en cambio, cada tipo de dependiente y cada tipo de precedente tiene un "manejador" registrado en el Gestor de dependencias y el Gestor de dependencias llama a estos manejadores para realizar el proceso específico de negocio adecuado al tipo, por ejemplo para decodificar un precedente o dependiente en una descripción legible por las personas o para volver a calcular un dependiente cuando sea necesario.

El almacenamiento de dependencias es opcional: Es importante tener en cuenta que la utilización del Gestor de dependencias para almacenar registros de dependencias es opcional.

Los clientes del Gestor de dependencias pueden elegir si los registros de dependencias son necesarios o no; es decir, si el cliente necesita la capacidad del Gestor de dependencias para identificar y volver a calcular dependientes automáticamente.

Por ejemplo, el motor de elegibilidad y titularidad utiliza el Gestor de dependencias para almacenar registros de dependencias para determinaciones de evaluación de caso (es decir, determinaciones que suelen producir pagos financieros y/o facturas). El motor de elegibilidad y titularidad necesita que el Gestor de dependencias informe cuando un caso debe reevaluarse, que es el motivo por el cual se deben almacenar los registros de dependencias.

Por el contrario, el motor de elegibilidad y titularidad también contiene una característica para que un asistente social compruebe manualmente la elegibilidad y titularidad de un caso, basándose en la prueba en edición. Estos cálculos manuales de elegibilidad/titularidad utilizan los mismos métodos de cálculo pero que no necesitan ningún almacenamiento de dependencias, ya que el sistema no

tiene que volver a calcular nunca dichas determinaciones; en lugar de ello, los cálculos manuales de elegibilidad/titularidad los desencadena siempre una solicitud explícita de un asistente social.

Granularidad de dependencias: Tenga en cuenta que los elementos de datos de precedente utilizados en el ejemplo anterior son deliberadamente vagos: el términos "detalles personales" cubre con toda probabilidad muchos campos de datos individuales como fechas de nacimiento/defunción, demografía, etc. El Gestor de dependencias no conoce ni valora los significados de las dependencias que almacena entre dependientes y precedentes: es decisión de los clientes del Gestor de dependencias darles significado y almacenar las dependencias en una granularidad apropiada.

La elección de granularidad implica encontrar un equilibrio aceptable entre los dos extremos de:

- **Granularidad muy precisa**

  Las dependencias muy precisas se almacenan entre los dependientes y los campos de datos individuales, permitiendo una identificación extremadamente ajustada de los dependientes afectados por los cambios de precedente, pero con un coste muy alto de registros de dependencia almacenados; frente a
- **Granularidad muy gruesa**

  Se almacenan dependencias muy gruesas entre dependientes y agrupaciones de grandes números de campos de datos individuales en un "elemento de datos", lo que hace que se almacene un pequeño número de registros de dependencias, con el riesgo de que se soliciten recálculos falsos (es decir, recálculos que terminan no siendo necesarios porque el cálculo no queda afectado por el campo de datos concreto que ha cambiado).

Es responsabilidad de los diseñadores de clientes del Gestor de dependencias tener en cuenta estas concesiones y tomar decisiones sensatas respecto al nivel en el que se debe almacenar la información de dependencias en el Gestor de dependencias.

Por ejemplo, supongamos que el sistema registra estos detalles personales acerca de un demandante (en un sistema realista pueden existir muchos más campos considerados como "detalles personales"):

- fecha de nacimiento (se utiliza en cálculos de titularidad);
- número de hijos (se utiliza en cálculos de titularidad) y
- apellido de la madre (la respuesta a una pregunta de seguridad, sólo se utiliza para confirmar la identidad del demandante; no se utiliza en cálculos de titularidad).

Un conjunto muy preciso de dependencias muestra que la titularidad de un caso depende de la fecha de nacimiento y del número de hijos, pero no del apellido de la madre (ya que no se ha accedido a él durante los cálculos):

Tabla 5. Ejemplo de matriz de dependencia precisa

Precedente	**Titularidad del caso 127**
Fecha de nacimiento de Fran	X
Número de hijos de Fran	X
Apellido de la madre de Fran	

Este almacenamiento de dependencia preciso puede terminar necesitando que se almacenen muchas filas; pero tenga en cuenta que sólo los cambios en la fecha de nacimiento y/o el número de hijos desencadenará un recálculo de la titularidad del caso; si sólo se corrige un error tipográfico en el apellido de la madre, no se desencadenará ningún recálculo de titularidad de caso.

Por el contrario, un conjunto muy general de dependencias mostrará un registro mucho más simple de que la titularidad del caso depende de los detalles personales generales:

Tabla 6. Ejemplo de matriz de dependencia general

Precedente	**Titularidad del caso 127**
Detalles personales de Fran	X

Este almacenamiento de dependencias general almacena menos registros de dependencias pero si se corrige un error tipográfico en el apellido de la madre madre, los detalles personales generales cambiarán y se desencadenará un recálculo de la titularidad del caso, aunque el recálculo muestre que el resultado del cálculo no ha cambiado.

Captura de elementos de cambio de precedente

Los clientes deben notificar al Gestor de dependencias siempre que el valor de un precedente ha cambiado. El Gestor de dependencias acumula estos cambios en un conjunto de cambios de precedente para procesarlos posteriormente.

El ejemplo más común es el motor de elegibilidad y titularidad, que contiene "propagadores de objeto de regla": estos propagadores son responsables de escuchas los cambios en las entidades y las pruebas y de notificar estos cambios al gestor de dependencias.

El Gestor de dependencias soporta estas modalidades para tratar con los cambios de precedente:

- Cola para proceso aplazado (valor predeterminado) y
- Cola para proceso por lotes (para uso por parte de clientes que identifican los cambios de precedente que probablemente harán que se vuelva a calcular un gran número de dependientes).

Estas modalidades se describen con más detalles en las secciones siguientes.

Cola para proceso aplazado: Esta es la modalidad predeterminada para manejar cambios de precedente.

Si se identifican elementos de cambio de precedente durante el ámbito de una transacción de base de datos, el Gestor de dependencias:

- crea un solo conjunto de cambios de precedente nuevo en la base de datos;
- añade todos los elementos de cambio de precedente identificados durante la transacción a este conjunto de cambios de precedente nuevo y
- pone en cola una solicitud de proceso aplazado para procesar este nuevo conjunto de cambios de precedente.

Cola para el proceso por lotes: Esta es una modalidad especial, utilizada sólo por clientes que identifican cambios en los valores de precedentes que probablemente hagan que se recalcule un gran número de dependientes. El Gestor de

dependencias mantiene un conjunto espacial de cambios de precedentes "por lotes" de todo el sistema que acumula cambios de precedente (potencialmente) de muchas transacciones diferentes.

Si se identifican elementos de cambio de precedente durante el ámbito de una transacción de base de datos, el Gestor de dependencias:

- recupera el conjunto especial de cambios de precedente por lotes de todo el sistema;
- añade todos los elementos de cambio de precedente identificados durante la transacción a este conjunto de cambios de precedente por lotes y
- graba un mensaje informativo en los registros de aplicación para solicitar a un administrador que planifique el proceso por lotes para procesar este conjunto de cambios de precedente por lotes.

Ejemplo: Cuando se actualizan los detalles personales de Juan en el sistema, el propagador de objetos de regla de entidad notifica al Gestor de dependencias del cambio y el Gestor de dependencias graba el cambio de precedente en un nuevo conjunto de cambios de precedente y pone en cola un proceso aplazado. El proceso aplazado identifica que los dos casos de Juan están potencialmente afectados y los revisa.

Posteriormente un administrador publica algunos cambios en los conjuntos de reglas CER. El gestor de dependencias registrar estos cambios en los conjuntos de reglas CER añadiendo un registro de elemento de cambio de precedente al conjunto de cambios de precedente de proceso por lotes de todo el sistema. El administrador también publica algunos cambios en las tarifas y el Gestor de dependencias registra estos cambios en las tarifas añadiendo otro registro de elementos de cambio de precedente al conjunto de cambios de precedente de proceso por lotes de todo el sistema. El administrador acuerda ejecutar la suite por lotes del Gestor de dependencias para identificar y reevaluar los casos afectados.

Ciclo de vida de un conjunto de cambios de precedente: Cada conjunto de cambios de precedente pasa por un ciclo de vida simple como se indica a continuación:

- **Abierto**

  Estado de un conjunto de cambios de precedente cuando se crea inicialmente. En este estado se pueden añadir elementos de cambio de precedente nuevos al conjunto de cambios de precedente.
- **Enviado**

  El conjunto de cambios de precedente se ha enviado a proceso de identificación de dependientes. No se pueden añadir más elementos de cambio de precedente al conjunto de cambios de precedente.
- **Completo**

  Se ha completado el recálculo de todos los dependientes afectados por los cambios de precedente. El conjunto de cambios de precedente sólo se conserva para fines históricos.

Las transiciones entre cada estado se producen de forma distinta en función de la modalidad en la que se hayan capturado los cambios de precedente:

- Cola para proceso diferido:
  - la transacción que captura los cambios de precedente *abre* un conjunto de cambios de precedente nuevo y lo *envía* solicitando un proceso aplazado y

  - el proceso aplazado acepta el conjunto de cambios de precedente *enviado*, identifica y recalcula los dependientes afectados y *completa* el conjunto de cambios de precedente.
- Cola para proceso por lotes:
  - la transacción que captura los cambios de precedente los graba en el conjunto de cambios de precedente de proceso por lotes *abierto* actualmente;
  - la suite de procesos por lotes del Gestor de dependencias realiza los pasos siguientes:
    - recupera el conjunto de cambios precedente del proceso por lotes actualmente *abierto* y lo *envía* al siguiente paso del proceso por lotes y crea un nuevo conjunto de cambios precedente del proceso por lotes *abierto* para capturar cualquier cambio precedente adicional identificado;

      Nota: En un sistema en ejecución, esto es cómo se crea un nuevo conjunto de cambios precedente de proceso por lotes, es decir, mediante su procesador que se está enviando. El conjunto de cambios precedente abierto *inicial* se proporciona mediante un archivo DMX incluido en la aplicación.
    - realiza el proceso por lotes en modalidad continua para identificar y recalcular los dependientes afectados por los cambios en el conjunto de cambios de precedente de proceso por lotes *sometido* ahora y
    - *completa* el conjunto de cambios de precedente de proceso por lotes.

Identificación de dependientes potencialmente afectados

Una vez que el Gestor de dependencias ha capturado uno o más elementos de cambio de precedente y los ha agrupado en un conjunto de cambios de precedente, el Gestor de dependencias puede identificar qué dependientes están potencialmente afectados por uno o varios de los elementos de cambio de precedente, examinando los registros de dependencia almacenados para cada precedente en el conjunto de cambios de precedente. Es en este punto donde el Gestor de dependencias filtra los dependientes que se identifican más de una vez.

Esta identificación de los dependientes potencialmente afectados se produce en proceso aplazado o en proceso por lotes, en función de la modalidad en vigor cuando los elementos de cambio de precedente fueron capturados (consulte "Captura de elementos de cambio de precedente" en la página 87).

Por ejemplo, si una transacción de base de datos única graba cambios en varias filas de base de datos, el paso de proceso aplazado identificará cada caso afectado sólo una vez, aunque cada fila de base de datos cambiada sola afecte a un caso concreto.

Del mismo modo, si el conjunto de cambios de precedente de proceso por lotes contiene cambios en los conjuntos de reglas CER y las tarifas, el paso de proceso por lotes volverá a identificar cada caso afectado sólo una vez, aunque el cambio de conjunto de reglas CER por sí solo afecte a un caso concreto, como lo hace también el cambio de tarifa por sí solo.

Recálculo de dependientes identificados

Una vez que el Gestor de dependencias ha identificado todos los dependientes que están potencialmente afectados por los cambios de precedente, el Gestor de dependencias solicita que se recalcule cada uno de esos dependientes. El Gestor de dependencias no conoce qué representa cada dependiente, por lo que para llevar a cabo el recálculo adecuado el Gestor de dependencias busca un manejador registrado para cada dependiente y delega la responsabilidad del recálculo al manejador.

Por ejemplo, el manejador de dependientes registrado para las determinaciones de valoración de casos comprende que el recálculo adecuado para un caso consiste en volver a evaluar el caso.

Este recálculo de un dependiente se produce en el proceso aplazado o en el proceso por lotes, en función de la modalidad en vigor cuando se capturaron los elementos de cambio precedentes (consulte "Captura de elementos de cambio de precedente" en la página 87). Una vez que este paso se ha completado, el sistema está actualizado respecto a los cambios de precedente que han sido capturados.

Nota: Puesto que se puede llamar a cada manejador de dependientes desde transacciones en línea, aplazadas o de proceso por lotes, es obligatorio que ningún manejador de dependientes *no* realice presunciones sobre el tipo de transacción en vigor.

Proceso aplazado del Gestor de dependencias

En la sección anterior ("Funciones del Gestor de dependencias" en la página 83) se describe cómo soporta el Gestor de dependencias la captura de elementos de cambio de precedente en un conjunto de cambios de precedente por lotes de todo el sistema y cómo el Gestor de dependencias contiene procesamiento aplazado para identificar los dependientes potencialmente afectados y volver a calcularlos.

En esta sección se proporcionan más detalles sobre el proceso aplazado del Gestor de dependencias.

El procesamiento aplazado precalculará inicialmente el número total de dependientes potencialmente afectados y, por tanto, cuántos recálculos deberá llevar a cabo y tomar así una de las dos decisiones siguientes:

- Si este cálculo está por debajo del límite del sistema, el proceso aplazado realizará los recálculos del conjunto de cambios en precedentes en el momento; o bien
- Si este cálculo sobrepasa un límite del sistema, los recálculos del conjunto de cambios en precedentes se moverán al proceso por lotes del gestor de dependencias ("Proceso por lotes del Gestor de dependencias" en la página 91). En este punto, el conjunto de cambios en precedentes se marcará con un estado de 'Aplazado a lote'.

Establecimiento del límite del sistema del proceso aplazado

El límite del sistema para procesos aplazados puede configurarse mediante la propiedad de aplicación `curam.dependency.deferred.processing.limit`.

Aunque el valor por omisión se establece a `50`, debe tenerse en cuenta que la cifra verdadera de un sistema dependerá de múltiples factores, tales como la memoria disponible. Por tanto, se recomienda que el valor se configure a partir de pruebas efectuadas en el sistema a fin de dar con un valor adecuado que pueda procesar la mayoría de los procesos aplazados, cuando no todos ellos.

Manejo de errores en el proceso aplazado

Si se produce un error durante la ejecución del proceso aplazado del gestor de dependencias tras el número habitual de reintentos, el sistema moverá el trabajo a un proceso por lotes en su lugar. Esto ofrece una protección frente a errores derivados de agotamientos del tiempo de espera de las transacciones, y garantiza que se intenten todas las vías posibles del procesamiento de los cálculos.

Además, el estado del conjunto de cambios en precedentes en este caso cambiará a 'Aplazado a lote' para indicar que el proceso aplazado dio un error, y se generará una notificación al originador del proceso aplazado.

Proceso por lotes del Gestor de dependencias

En una sección anterior (“Funciones del Gestor de dependencias” en la página 83) se describía cómo soporta el Gestor de dependencias la captura de elementos de cambio de precedente en un conjunto de cambios de precedente por lotes de todo el sistema y cómo el Gestor de dependencias contiene el proceso por lotes para identificar los dependientes potencialmente afectados y volver a calcularlos.

En esta sección se proporcionan más detalles sobre el proceso por lotes del Gestor de dependencias.

El Gestor de dependencias mantiene registros de control en la base de datos para apuntar a los conjuntos de cambios de precedentes por lotes siguientes:

- el conjunto de cambios de precedentes de proceso por lotes actualmente abierto (siempre habrá exactamente un conjunto de cambios de precedente de proceso por lotes que esté abierto para aceptar elementos de cambio de precedente nuevos) y
- el conjunto de cambios de precedente de proceso por lotes procesado actualmente por la suite de proceso por lotes del Gestor de dependencias (si existe - sólo se llena durante el proceso por lotes; la mayoría de las veces no hay ningún conjunto de cambios de precedente de proceso por lotes en este estado).

Estos registros de control son fundamentales para el comportamiento de la suite de proceso por lotes del Gestor de dependencias.

Siempre que hay cambios precedentes en la modalidad de "cola para el proceso por lotes", los registros de la aplicación mostrarán un mensaje que notifica al administrador que es necesaria una ejecución de la suite del proceso por lotes del gestor de dependencias. El usuario que ha realizado los cambios que estaban en cola para el proceso por lotes también recibirá un mensaje informativo en pantalla que advierte que es necesaria una ejecución de la suite de proceso por lotes del gestor de dependencias.

La suite de proceso por lotes del Gestor de dependencias está compuesta de estos procesos por lotes independientes:

- **Enviar conjunto de cambios de precedente**

  El punto de partida de la suite de proceso por lotes. Proceso ligero de una sola corriente que envía el conjunto de cambios de precedente de proceso por lotes abierto actualmente.
- **Realizar recálculos por lotes desde el conjunto de cambios de procedente**

  Proceso pesado de varias corrientes que identifica los dependientes que están potencialmente afectados por los cambios en el conjunto de cambios de precedente enviado y los recalcula. El tiempo que se tarda en ejecutar este proceso variará en función de cuántos recálculos de dependientes sean necesarios y puede ser considerable.
- **Completar conjunto de cambios de precedente**

  Punto final de la suite de proceso por lotes. Proceso ligero de una sola corriente que completa el conjunto de cambios de precedente de proceso por lotes enviado actualmente.

Estos procesos por lotes se describen más detalladamente en las secciones siguientes.

Enviar conjunto de cambios de precedente

Este proceso por lotes es el punto de partida para la suite de proceso por lotes, que contiene un proceso ligero de una solo corriente que envía el conjunto de cambios de precedente de proceso por lotes actualmente abierto y crea un nuevo conjunto de cambios de precedente de proceso por lotes abierto, que se utilizará para capturar los cambios de precedente subsiguientes identificados y puestos en cola para su proceso por lotes.

Para ejecutar este proceso por lotes, ejecute el mandato siguiente (en una sola línea):

```
build runbatch -Dbatch.program=

curam.dependency.intf.SubmitPrecedentChangeSet.process

-Dbatch.username=SYSTEM
```

Si este proceso por lotes se completa satisfactoriamente, producirá un mensaje simple confirmando que el conjunto de cambios de precedente de proceso por lotes abierto se ha enviado.

Consejo: Un error común consiste en intentar ejecutar este proceso por lotes cuando otro conjunto de cambios de precedente de proceso por lotes está todavía en el estado enviado.

Este proceso producirá un mensaje de error simple si hay otro conjunto de cambios de precedente de proceso por lotes de este tipo que aún no ha sido procesado completamente por la suite de proceso por lotes.

Por comodidad, este proceso por lotes también genera una lista de los tipos de dependiente que están registrados en el Gestor de dependencias; esta lista de tipos de dependiente es importante cuando se ejecuta el paso siguiente ("Realizar recálculos por lotes en el conjunto de cambios de precedentes" en la página 93).

Los tipos de dependiente que se incluye con la aplicación son:

- **Resultado de la determinación de la evaluación del caso**

  El cálculo de una determinación de evaluación para un caso de entrega de producto.

  Consulte la guía `Inside Cúram Eligibility and Entitlement Using Cúram Express Rules`.
- **Contexto de asesoramiento**

  El cálculo de consejos.

  Consulte la `Guía de configuración de asesor de Cúram`.
- **Valor de atributo almacenado**

  El cálculo de un atributo almacenado en las tablas de base de datos de CER.

  Consulte "CER utiliza el Gestor de dependencias para almacenar dependencias para los valores de atributos almacenados por CER" en la página 102.

Realizar recálculos por lotes en el conjunto de cambios de precedentes

Este proceso por lotes es el proceso pesado de varias secuencias que identifica los dependientes potencialmente afectados por los cambios del conjunto de cambios precedente enviado y los vuelve a recalcular. El tiempo que se tarda en ejecutar este proceso varía en función de cuántos recálculos dependientes son necesarios y se pueden considerar.

El paso Realizar recálculos por lotes a partir del conjunto de cambios precedente se debe ejecutar varias veces, una para cada tipo de dependiente registrado con el gestor de dependencias (consulte la salida que se genera mediante el paso anterior "Enviar conjunto de cambios de precedente" en la página 92). Es libre de elegir el orden más apropiado en el que procesar los tipos de dependiente; por ejemplo, es probable que sea más importante para la empresa reevaluar determinaciones de caso (consulte la guía `Inside Cúram Eligibility and Entitlement Using Cúram Express Rules`) que identificar asesoramiento anticuado (consulte la `Guía de configuración de asesor`). También puede difundir el proceso para distintos tipos de dependientes durante varios días, pero tenga en cuenta que los elementos de cambio de precedente adicionales en cola para proceso por lotes no se pueden procesar hasta que el conjunto de cambios de precedente sometido actualmente sometido haya completado la suite de proceso por lotes de gestor de dependencias completa.

El paso Realizar recálculos por lotes a partir del conjunto de cambios precedente utiliza la arquitectura de modalidad continua de proceso por lotes de Cúram (consulte la guía *Cúram Batch Performance Mechanisms Guide*) y, como tal, el proceso se divide en estas fases>

- **Identificar trozos**

  Fase, que debe ejecutarse como un proceso único, que identifica a los dependientes (de un tipo determinado de dependiente) potencialmente afectados por los cambios en el conjunto de cambios de precedente de proceso por lotes enviado. Los ID de los dependientes identificado se graban en "trozos" para que los procese la siguiente fase.
- **Procesar trozos**

  Fase, apta para la ejecución simultánea por parte de varios procesos, que toma un trozo de dependientes identificados y recalcula cada dependiente.

Para ejecutar este proceso por lotes para un tipo de dependiente particular, emita el mandato siguiente en una línea:

```
build runbatch -Dbatch.program=
curam.dependency.intf.PerformBatchRecalculationsFromPrecedentChangeSet.process
-Dbatch.username=SYSTEM
 -Dbatch.parameters="dependentType= code-for-dependent-type "
```

De forma predeterminada, el único proceso realiza ambas fases; sin embargo, puede ejecutar procesos adicionales de "Secuencia" de forma simultánea en otros sistemas para realizar la segunda fase en paralelo (Si desea más información sobre el proceso paralelo y las variables de entorno que rigen el comportamiento del proceso paralelo de este proceso Realizar recálculos por lotes a partir del conjunto de cambios precedente, consulte la guía *Cúram Batch Performance Mechanisms Guide*. Para ejecutar un proceso de "secuencia" para un tipo de dependiente particular, ejecute el mandato siguiente en una línea:

```
build runbatch -Dbatch.program=
curam.dependency.intf.PerformBatchRecalculationsFromPrecedentChangeSetStream.process
-Dbatch.username=SYSTEM
 -Dbatch.parameters="dependentType= code-for-dependent-type "
```

El proceso por lotes no se puede iniciar con un error no recuperable si se produce algo de lo siguiente:

- el dependentType no se proporciona, o se proporciona pero no es el código para ningún tipo de dependiente registrado con el gestor de dependencias; o
- no hay ningún conjunto de cambios precedente por lotes en el estado "enviado" (es decir, el proceso Enviar conjunto de cambios precedente todavía no se ha ejecutado desde la última ejecución de Completar conjunto de cambios precedente).

De lo contrario, se inicia el proceso por lotes y se intenta identificar y recalcular los dependientes afectados. El resultado de intentar recalcular un dependiente particular es:

- **Correcto**

  El dependiente se ha encontrado y se ha vuelto a calcular correctamente y el proceso continúa con normalidad.
- **No encontrado**

  El dependiente no se ha encontrado y, por lo tanto, no se ha podido procesar. Esta situación puede producirse si un cliente del Gestor de dependencias decide que un dependiente ya no debe existir, pero no solicita al Gestor de dependencias que elimine los registros de dependencia de ese dependiente. En estas circunstancias, el Gestor de dependencias elimina automáticamente los registros de dependencia externos y graba un aviso en la salida de corriente de registro/proceso por lotes de la aplicación.
- **Error**

  Se ha lanzado una excepción durante el recálculo del dependiente. Por ejemplo, si un cálculo de CER ha encontrado un problema "división por cero". La excepción generada se graba en la salida de corriente de proceso por lotes y el proceso de recuperación lo maneja el proceso "saltar" de la Arquitectura de modalidad continua de proceso por lotes de Cúram.

Cuando acaba el proceso Realizar recálculos por lotes a partir del conjunto de cambios precedente, se escribe un informe global con detalles de cuántos dependientes se han procesado correctamente, en relación con los no encontrados y los errores encontrados. Si se ha encontrado algún error, examine los registros de salida de las corrientes por lotes para obtener detalles de los errores.

Importante: Si define el nivel de registro estándar de Cúram en "verboso" o superior, antes de volver a calcular cada dependiente, el gestor de dependientes da como salida:

- una descripción legible por las personas del dependiente y
- el subconjunto de cambios de precedente (del conjunto de cambios de precedente) que ha hecho que se identificara el dependiente.

Este nivel de registro es apto solo para entornos de desarrollo. El registro detallado puede afectar negativamente al rendimiento y a la escalabilidad en un sistema de producción.

Esta salida puede ser útil para comprender por qué se ha identificado un dependiente particular como que requiere un recálculo.

Nota: Si por accidente ejecuta este proceso por lotes más de una vez para el mismo tipo de dependiente, se producen recálculos de los dependientes, pero el recálculo encuentra que el dependiente ya está actualizado.

Como tal, este tipo de ejecución adicional fortuita para un tipo de dependiente no daña el sistema, pero malgasta un tiempo valioso de proceso.

Tome nota con atención de la lista de tipos de dependientes y rastree qué tipos de dependiente procesa y qué tipos de dependiente quedan por procesar.

Completar conjunto de cambios de precedente

El punto final para la suite de proceso por lotes, que contiene un proceso ligero de una sola corriente que completa el conjunto de cambios de precedente de proceso por lotes sometido actualmente.

PRECAUCIÓN:
No ejecute este proceso por lotes hasta que esté seguro de que el paso anterior ("Realizar recálculos por lotes en el conjunto de cambios de precedentes" en la página 93) ha finalizado *para cada tipo dependiente registrado en el Gestor de dependencias***.**

Para ejecutar este proceso por lotes, ejecute el mandato siguiente (en una sola línea):

```
build runbatch -Dbatch.program=

curam.dependency.intf.CompletePrecedentChangeSet.process

-Dbatch.username=SYSTEM
```

Si este proceso por lotes se completa satisfactoriamente, producirá un mensaje simple confirmando que el conjunto de cambios de precedente de proceso por lotes sometido se ha completado.

Consejo: Un error común es intentar ejecutar este proceso por lotes antes de que se haya ejecutado el sometimiento del conjunto de cambios de precedente.

Este proceso producirá un mensaje de error simple si no hay ningún conjunto de cambios de precedente de proceso por lotes en estado "sometido".

Después de que se haya completado el conjunto de cambios de precedente de proceso por lotes, el proceso comprueba si los cambios de precedente nuevos se han puesto en cola para proceso por lotes adicional desde que empezó la suite de proceso por lotes. Esta situación puede producirse si

- el recálculo de cualquier dependiente durante la ejecución por lotes ha dado lugar a cambios en los datos que también se utilizan como precedente y/o
- el sistema en línea se ha estado ejecutando simultáneamente con la suite de lotes y un usuario ha realizado cambios que han hecho que los elementos de cambio de precedente se pusieran en cola para el proceso por lotes.

El proceso se producirá un mensaje simple informando de que:

- no hay elementos de cambio de precedente adicionales en cola para el proceso por lotes (y, por lo tanto, el sistema está actualizado respecto a los elementos de cambio de precedente por lotes) o

- se han puesto en cola elementos de cambio de precedente adicionales para el proceso por lotes y se necesita otra ejecución de la suite de proceso por lotes del Gestor de dependencias para procesar estos elementos adicionales. En esta situación, tendrá que decidir si va a ejecutar la ejecución adicional inmediatamente o esperar hasta un momento posterior (quizás cuando se hayan puesto en cola incluso más elementos de cambio de precedente de proceso por lotes para el proceso por lotes).

Herramientas de proceso por lotes del gestor de dependencias

si desea información sobre las herramientas proporcionadas para ayudar a la ejecución de la suite del proceso por lotes del gestor de dependencias, consulte la sección 'Herramientas de proceso por lotes del gestor de dependencias' de la 'Guía de operaciones Cúram'.

Integración entre CER y el Gestor de dependencias

CER se integra con el Gestor de dependencias de varias maneras importantes:

- CER puede identificar dependencias para que las almacenen los clientes del Gestor de dependencias;
- CER utiliza el Gestor de dependencias para almacenar dependencias para cualquier valor de atributo calculado almacenado en las tablas de base de datos de CER y
- El Gestor de dependencias solicita a CER que recalcule los valores de atributo calculados almacenados en las tablas de base de datos de CER si cambia alguno de los precedentes.

Estos puntos de integración entre CER y el Gestor de dependencias se describen con más detalle en las secciones siguientes.

Programa de utilidad CER para identificar dependencias que se deben almacenar

Un cliente de CER utiliza CER para realizar cálculos complejos. Normalmente el cliente de CER también puede necesitar almacenar dependencias en el Gestor de dependencias de forma que el Gestor de dependencias pueda notificar al cliente siempre que cambien los precedentes y ese cliente pueda entonces volver a invocar CER para volver a calcular la salida, teniendo en cuenta los cambios en los datos de precedente.

CER contiene un programa de utilidad para ayudar a sus clientes a identificar las dependencias a almacenar en el Gestor de dependencias. El programa de utilidad toma un valor de atributo (que CER ha calculado) y devuelve un conjunto de precedentes para ese valor de atributo. Entonces un cliente de CER pasa su dependiente y los precedentes identificados al Gestor de dependencias para almacenar los registros de dependencias.

Cuando CER calcula un valor de atributo, mantiene en la memoria un árbol completo de dependencias lógicas que contiene:

- como raíz, el propio valor de atributo calculado;
- como nodos de bifurcación, los valores de cálculo intermedio (normalmente en objetos de regla interna);
- como nodos hoja, los datos de entrada externos recuperados durante el cálculo de CER.

El programa de utilidad es capaz de analizar este árbol de dependencias lógicas para proporcionar un conjunto mucho más pequeño de precedentes, normalmente

basándose en los nodos hoja del árbol; es decir, normalmente los resultados de cálculo intermedios se ignoran y las dependencias almacenadas reflejan que el valor de atributo calculado depende finalmente de los datos de entrada externos a los que se accede durante los cálculos.

Nota: Cualquier cálculo no trivial, como aquellos normalmente realizados por CER, tendrá varios valores derivados intermedios "entre" el dependiente general y los precedentes de entrada.

Estos valores intermedios no se pasan al Gestor de dependencias; en lugar de ello, el gestor de dependencias almacena registros de dependencia que enlazan dependientes de alto nivel (como titularidad de un caso) directamente a sus precedentes de bajo nivel (como datos de entidad, pruebas y velocidad).

Los valores intermedios no son de interés cuando se almacenan dependencias.

Los precedentes identificados por el programa de utilidad son una mezcla de:

- precedentes identificados directamente por CER y
- precedentes identificados por los conversores de objeto de regla registrados en el almacenamiento de datos de base de datos de CER.

Precedentes identificados directamente por CER:

El programa de utilidad identifica directamente estos tipos de dependencias para un valor de atributo calculado.

El cálculo general es el cálculo del propio atributo, o de cualquiera de los valores de atributo interno del que depende básicamente (los cálculos "intermedios" entre el valor de atributo calculado y sus entradas de datos externos).

Tabla 7. Precedentes identificados directamente por CER

Nombre	Cuándo se identifica	Desencadenante para recálculo
Valor de atributo almacenado	Identifica los valores de atributo "de entrada" almacenados en las tablas de base de datos de CER que se recuperó durante el cálculo general del valor del atributo. El ID de precedente hace referencia al ID interno para la fila de base de datos del atributo almacenado en las tablas de base de datos de CER.	Si el valor del atributo almacenado cambia, se grabará un elemento de cambio de precedente para el valor de atributo almacenado se grabará en un conjunto de cambios de precedente.
Definiciones de conjunto de reglas	Identifica cada conjunto de reglas que contiene cualquiera de los atributos utilizados en el cálculo general del valor de atributo. El ID de precedente hace referencia al nombre del conjunto de reglas que contiene una o más definiciones de atributo encontradas durante el cálculo general.	Si se publican los cambios realizados en un conjunto de reglas CER, se grabará en un conjunto de cambios de precedente un elemento de cambio de precedente para el conjunto de reglas cambiado.

Tabla 7. Precedentes identificados directamente por CER (continuación)

Nombre	Cuándo se identifica	Desencadenante para recálculo
Búsqueda de 'readall'	Identifica las expresiones "readall" en la página 216 (sin match anidado) encontradas durante el cálculo general, que han recuperado objetos de regla almacenados en tablas de base de datos de CER (en lugar de objetos de regla recuperados utilizando conversores de objetos de regla - en su lugar, consulte "Precedentes identificados por conversiones de objeto de regla" en la página 102). El ID de precedente hace referencia al nombre de la clase de regla buscado por la expresión "readall" en la página 216.	Un elemento de cambio de precedente para la clase de regla se grabará en un conjunto de cambios de precedente si: • Un nuevo objeto de regla que dicha clase de regla se almacena en las tablas de la base de datos de CER y/o • Un objeto de regla existente para dicha clase de regla se elimina de las tablas de la base de datos de CER.
Búsqueda de 'readall/match'	Identifica las expresiones "readall" en la página 216 encontradas durante el cálculo general, que han recuperado objetos de regla almacenados en tablas de base de datos de CER (en lugar de objetos de regla recuperados utilizando conversores de objeto de regla - en su lugar, consulte "Precedentes identificados por conversiones de objeto de regla" en la página 102). El ID de precedente hace referencia al nombre de la clase de regla buscada por la expresión "readall" en la página 216, junto con el nombre de atributo y valor utilizados como criterios de búsqueda.	Un elemento de cambio de precedente para la clase de regla y el nombre de atributo y valor de coincidencia se grabarán en un conjunto de cambios de precedente si: • Un nuevo objeto de regla para dicha clase de regla se almacena en tablas de base de datos de CER; • Un objeto de regla existente para dicha clase de regla se elimina de las tablas de base de datos de CER; y/O • El valor del atributo utilizado en el criterio de búsqueda cambia para un objeto de regla existente (en cuyo caso se escribirán dos elementos de cambio precedentes, uno para el valor antiguo del atributo y otro para el nuevo valor del atributo).

Estos tipos de dependencias se ilustran mejor con un ejemplo.

Supongamos que se graba un sistema nuevo que utiliza CER para calcular la responsabilidad fiscal de una persona. Este sistema de Responsabilidad fiscal utiliza el Gestor de dependencias para almacenar dependencias, para que la responsabilidad fiscal se pueda volver a calcular (utilizando CER) si cambian las circunstancias de la persona.

El sistema de responsabilidad fiscal almacena información de "umbral de impuestos" de todo el sistema en objetos de regla, con estos objetos de regla almacenados en tablas de base de datos de CER. Las reglas CER para calcular la responsabilidad fiscal de una persona incluyen una expresión "readall" en la página 216 para recuperar todos los umbrales de impuestos del sistema.

El sistema de responsabilidad fiscal también almacena información de "activos" de todo el sistema en objetos de regla, con estos objetos de regla almacenados en tablas de base de datos de CER. Cada activo especifica su propietario y su valor de mercado. Las reglas CER para calcular la responsabilidad fiscal de una persona incluyen una expresión "`readall`" en la página 216 para recuperar todos los activos que son propiedad de dicha persona (es decir, aquellos con un `Asset.ownedByPersonID` coincidente con `Person.personID`). Es posible que el valor de mercado de un activo cambie y/o que un activo se transfiera de una persona a otra, cambiando su `Asset.ownedByPersonID` de un ID de persona a otro.

Las reglas CER para calcular la responsabilidad fiscal de una persona incluyen la suma de `Asset.marketValue` de todos los activos que son propiedad de dicha persona.

El sistema de responsabilidad fiscal contiene conjuntos de reglas CER independientes para recuperar los datos de entrada necesarios para el cálculo de responsabilidad fiscal frente a los cálculos de negocio actuales que utilizan los datos recuperados para calcular la responsabilidad fiscal de una persona.

Un usuario utiliza el sistema de responsabilidad fiscal para calcular la responsabilidad fiscal de Juan (personid 456) y de María (personid 457), cada uno de los cuales tienen un activo. El sistema de responsabilidad fiscal utiliza el programa de utilidad CER para identificar las dependencias y las pasa al gestor de dependencias para su almacenamiento, lo que hace que se almacenen las dependencias siguientes:

Tabla 8. Ejemplo de Dependencias almacenadas para responsabilidad fiscal

Tipo de dependiente	**ID de dependiente**	**Tipo de precedente**	**ID de precedente**
Responsabilidad fiscal	456 (ID de persona de Juan)	Búsqueda de 'readall'	Clase de regla: TaxThreshold Se almacena porque el cálculo de la responsabilidad fiscal de Juan ha producido una recuperación de todos los objetos de regla TaxThreshold.
Responsabilidad fiscal	456	Búsqueda de 'readall/ match'	Clase de regla: Activo, con el valor de atributo ownedByPersonID=456 Se almacena porque el cálculo de la responsabilidad fiscal de Juan ha producido una recuperación de todos los objetos de regla de activos que son propiedad de Juan.
Responsabilidad fiscal	456	Valor de atributo almacenado	789 (ID interno de `Asset.marketValue` para el activo de Juan) Se almacena porque el cálculo de la responsabilidad fiscal de Juan ha accedido al valor almacenado de marketValue en el único activo recuperado para Juan.

Tabla 8. Ejemplo de Dependencias almacenadas para responsabilidad fiscal (continuación)

Tipo de dependiente	ID de dependiente	Tipo de precedente	ID de precedente
Responsabilidad fiscal	456	Definiciones de conjunto de reglas	TaxLiabilityDataRetrievalRuleSet Se almacena porque el cálculo de la responsabilidad fiscal de Juan implicaba las definiciones de los atributos de regla de TaxLiabilityDataRetrievalRuleSet (se utiliza para recuperar los objetos de regla de TaxThreshold y Asset).
Responsabilidad fiscal	456	Definiciones de conjunto de reglas	TaxLiabilityBusinessCalculationsRuleSet Se almacena porque el cálculo de la responsabilidad fiscal de Juan implicaba las definiciones de los atributos de regla en TaxLiabilityBusinessCalculationsRuleSet (se utiliza para calcular la responsabilidad fiscal general basándose en los datos de entrada).
Responsabilidad fiscal	457 (ID de persona de María)	Búsqueda de 'readall'	Clase de regla: TaxThreshold Se almacena porque el cálculo de la responsabilidad fiscal de María ha producido una recuperación de todos los objetos de regla TaxThreshold.
Responsabilidad fiscal	457	Búsqueda de 'readall/ match'	Clase de regla: Activo, con el valor de atributo ownedByPersonID=457 Se almacena porque el cálculo de la responsabilidad fiscal de María ha producido una recuperación de todos los objetos de regla de activos que son propiedad de María.
Responsabilidad fiscal	457	Valor de atributo almacenado	780 (ID interno de `Asset.marketValue` para el activo de María) Se almacena porque el cálculo de la responsabilidad fiscal de María ha accedido al valor almacenado de marketValue en el único activo recuperado para María.
Responsabilidad fiscal	457	Definiciones de conjunto de reglas	TaxLiabilityDataRetrievalRuleSet Se almacena porque el cálculo de la responsabilidad fiscal de María implicaba las definiciones de los atributos de regla de TaxLiabilityDataRetrievalRuleSet (se utiliza para recuperar los objetos de regla de TaxThreshold y Asset).

Tabla 8. Ejemplo de Dependencias almacenadas para responsabilidad fiscal (continuación)

Tipo de dependiente	ID de dependiente	Tipo de precedente	ID de precedente
Responsabilidad fiscal	457	Definiciones de conjunto de reglas	TaxLiabilityBusinessCalculationsRuleSet Se almacena porque el cálculo de la responsabilidad fiscal de María implicaba las definiciones de los atributos de regla en TaxLiabilityBusinessCalculationsRuleSet (se utiliza para calcular la responsabilidad fiscal general basándose en los datos de entrada).

Ahora podemos ver cómo los diversos cambios en los datos desencadenarán recálculos de responsabilidad fiscal:

Tabla 9. Ejemplo de elementos de cambio de precedente para responsabilidad fiscal

Cambio de datos	Elementos de cambios de precedentes registrados	Recálculos desencadenados
El valor de mercado del activo de Juan aumenta de 100 $ a 120 $	• Valor de atributo almacenado, 789	• Se recalcula la responsabilidad fiscal de Juan
María vende el activo y el objeto de regla se elimina	• Búsqueda de 'readall/match', Clase de regla: Asset, con un valor de atributo ownedByPersonID=457	• Se recalcula la responsabilidad fiscal de María
Juan recibe un nuevo activo, almacenado como objeto de regla nuevo	• Búsqueda de 'readall/match', Clase de regla: Asset, con un valor de atributo ownedByPersonID=456	• Se recalcula la responsabilidad fiscal de Juan
Juan transfiere su primer activo a María, por lo tanto el ownedByPersonID del activo cambia de 456 a 457	• Búsqueda de 'readall/match', Clase de regla: Asset, con un valor de atributo ownedByPersonID=456 (valor antiguo) • Búsqueda de 'readall/match', Clase de regla: Asset, con el valor de atributo ownedByPersonID=457 (valor nuevo)	• Se recalcula la responsabilidad fiscal de Juan • Se recalcula la responsabilidad fiscal de María
Un administrador entra un umbral de impuestos nuevo, almacenado como objeto de regla nuevo	• Búsqueda de 'readall', Clase de regla: TaxThreshold	• Se recalcula la responsabilidad fiscal de Juan • Se recalcula la responsabilidad fiscal de María
Un administrador elimina un umbral de impuestos existente, eliminando así el objeto de regla	• Búsqueda de 'readall', Clase de regla: TaxThreshold	• Se recalcula la responsabilidad fiscal de Juan • Se recalcula la responsabilidad fiscal de María
Un administrador publica cambios en el conjunto de reglas TaxLiabilityBusinessCalculationsRuleSet	• Definiciones de conjunto de reglas, TaxLiabilityBusinessCalculationsRuleSet	• Se recalcula la responsabilidad fiscal de Juan • Se recalcula la responsabilidad fiscal de María

Precedentes identificados por conversiones de objeto de regla: Los conversores de objeto de regla registrados en el almacenamiento de datos de base de datos de CER también pueden contribuir en las dependencias identificadas por el programa de utilidad CER.

Consulte la documentación de cada conversor de objeto de regla para obtener detalles sobre las dependencias identificadas.

CER utiliza el Gestor de dependencias para almacenar dependencias para los valores de atributos almacenados por CER

En el caso de objetos de regla que están almacenados en las tablas de base de datos de CER, cada valor de atributo en esos objetos de regla puede desempeñar un rol como:

- dependiente: es decir, un valor de atributo que se deriva calculando los valores a partir de los datos de entrada y/o
- precedente: es decir, un valor de atributo que se utiliza como datos de entrada durante el cálculo de un dependiente.

CER registra un manejador de dependientes y un manejador de precedentes en el Gestor de dependencias para permitir que los valores de atributo almacenados se utilicen como dependientes y/o precedentes en las dependencias almacenadas.

Por ejemplo, si está presente un atributo `totalIncome` en un objeto de regla `Person` almacenado en tablas de base de datos de CER y el cálculo de `totalIncome` implicaba la recuperación de los valores de atributo `Income.amount` también almacenados en las tablas de base de datos de CER, CER solicitará al gestor de dependencias que almacene el hecho de que el valor de atributo `Person.totalIncome` depende de los valores de atributo `Income.amount`.

El Gestor de dependencias solicita a CER que recalcule los valores de atributos calculados almacenados

Cuando cambien los valores de precedente, el Gestor de dependencias identificará en CER los valores de atributo almacenados que dependen de esos valores de precedente cambiados y solicitará a CER que recalcule los valores de dependiente, a través del manejador de dependientes que CER registra en el Gestor de dependencias.

Por ejemplo, si un valor de atributo `Income.amount` almacenado en las tablas de base de datos de CER ha cambiado su valor, CER notificará al gestor de dependencias que el precedente `Income.amount` ha cambiado y el Gestor de dependencias identificará posteriormente que el dependiente `Person.totalIncome` necesita un recálculo e invocará CER para recalcularlo.

Conformidad

Consulte “Gestor de dependencias” en la página 264 para conocer la declaración de conformidad para el Gestor de dependencia.

Acerca del Editor CER

El Editor CER facilita la creación y el mantenimiento de conjuntos de reglas CER. Algunos conjuntos de reglas se basan en la legislación y otros ayudan a los usuarios a realizar determinadas actividades. Por lo tanto, el Editor CER tiene dos vistas básicas: la vista de empresa para aquellos que están familiarizados con la

estructura y el texto necesarios para la legislación y la vista técnica para aquellos usuarios familiarizados con los aspectos de implementación de desarrollo de las reglas.

El Editor CER contiene una barra de menú global que consta de características de usuario comunes como *Guardar*, *Buscar* y *Deshacer y Rehacer*. Existen otras opciones Guardar para permitir al usuario *Exportar*, *Validar* y *Guardar todo* los conjuntos de reglas.

Menú global de Editor CER

El Editor CER proporciona funcionalidad común como parte del menú global para facilitar el acceso.

Menú global de Editor CER

Tabla 10. Barra de menús

Nombre	Descripción
Deshacer	Para cancelar la última edición, elija deshacer en la barra de menús.
Rehacer	Para cancelar el último deshacer, elija Rehacer en la barra de menús; esto retrotraerá la última acción realizada.
Incluir conjuntos de reglas	Para ver los conjuntos de reglas incluidos y permitir añadir conjuntos de reglas proporcionando una vía de acceso de clases.
Exportar	El editor CER soporta la exportación de todos los diagramas de la vista de esquema de regla a imágenes PNG que se pueden guardar como un solo archivo de archivado ZIP en el disco duro local del usuario.
Guardar	Un conjunto de reglas se puede guardar en cualquier momento en que se haya abierto en el editor. Al guardar un conjunto de reglas que tiene un estado de publicado, se crea un registro de "Edición en curso" para ese conjunto de reglas. Las modificaciones que realice se guardan en el registro de edición en curso. Al guardar un conjunto de reglas que tiene un estado de "Edición en curso", se guardan las modificaciones directamente. Los conjuntos de reglas también pueden tener un estado de "Recién creado". Al guardar las modificaciones en un conjunto de reglas recién creado, las modificaciones se guardarán directamente en ese conjunto de reglas recién creado.
Guardar todo	El Editor CER soporta la apertura de varios conjuntos de reglas desde dentro de una instancia del editor. Varios elementos CER pueden apuntar a clases o atributos que se definen en los demás conjuntos de reglas. Los elementos CER que apuntan a algo en un conjunto de reglas diferente tendrán una opción de menú *Abrir conjunto de reglas*. Esto permite que un usuario tenga varios conjuntos de reglas abiertos en el editor en cualquier momento. La opción de menú *Guardar todo* guardará todos los conjuntos de reglas que están abiertos en el editor y que se han modificado.

Tabla 10. Barra de menús (continuación)

Nombre	Descripción
Validar	Permite al usuario validar los cambios en un conjunto de reglas. Se llama al validador de conjunto de reglas del motor de reglas CER. Los conjuntos de reglas CER son archivos XML que se ajustan al esquema de reglas proporcionado por CER. CER también incluye un validador de conjunto de reglas completo que puede detectar errores en el conjunto de reglas antes de permitir que se ejecute el conjunto de reglas. El validador de conjunto de reglas CER informa de una lista de errores en el panel Propiedades y validaciones para que el usuario los pueda resolver. Si hay errores, el validador de conjunto de reglas CER informa de los errores y el proceso se detiene.
Buscar	Permite al usuario realizar una búsqueda rápida basada en texto que abrirá una ventana emergente de resultados de búsqueda con resultados coincidentes. Se puede encontrar más información acerca de la funcionalidad proporcionada y cómo refinar una búsqueda en la sección de Herramientas de búsqueda.

Herramientas de búsqueda de Editor CER

Criterios de búsqueda de Editor CER: Un usuario puede realizar una búsqueda basada en texto general para los criterios siguientes (reglas y referencias de regla, descripciones, carpetas, datos técnicos y tablas de código):

Tabla 11. Criterios de búsqueda

Nombre	Descripción
Reglas y referencias de regla	Buscar regla proporcionando el nombre de la regla.
Descripciones	Buscar reglas o atributos en una regla entrando parte del texto utilizado para describir la regla o el atributo.
Carpetas	Buscar una carpeta proporcionando el nombre de la carpeta.
Datos técnicos	Buscar los atributos que se son atributos especificados o no especificados.
Tablas de códigos	Buscar los atributos que se especifican como tablas de códigos.

Vista de empresa

La Vista de empresa proporciona al usuario de empresa una vista centralizada de reglas donde están disponibles los elementos relevantes para crear y mantener las reglas empresariales. Consiste en un árbol o vista jerárquica de las reglas, lienzo de reglas, paletas de elementos de negocio y el panel de validación y propiedades.

Vista Esquema de reglas: Esto muestra todas las reglas de nivel superior y las carpetas en las que están ubicadas. Hay un menú contextual disponible en cada elemento dentro de la vista esquema:

Tabla 12. Nuevos elementos de menú

Nombre	Descripción
Carpeta (Botón)	Crear una carpeta proporcionando un nombre de carpeta nuevo.

Tabla 12. Nuevos elementos de menú (continuación)

Nombre	Descripción
Regla (Botón)	Crear una regla nueva proporcionando un nombre de regla nuevo y el tipo de datos para esta regla que tomará de forma predeterminada un tipo booleano. La regla se creará en la carpeta seleccionada. Si no hay ninguna carpeta seleccionada, la regla nueva se creará en el nivel raíz.
Nueva carpeta	Crear una carpeta proporcionando un nombre de carpeta nuevo.
Nueva regla	Crear una regla nueva proporcionando un nombre de regla nuevo y el tipo de datos para esta regla que tomará de forma predeterminada un tipo booleano. La regla se creará en la carpeta seleccionada. Si no hay ninguna carpeta seleccionada, la regla nueva se creará en el nivel raíz.
Suprimir	Suprimir una carpeta o regla según lo que esté resaltado en la vista esquema.
Establecer tipo de regla	Crear derivación para un atributo. Se proporciona una lista de tipos de datos para una regla nueva.
Mover regla	Esto permite al desarrollador de reglas mover una regla de una carpeta a otra carpeta seleccionando el nombre de carpeta.
Buscar referencias a esta regla	Permite al usuario buscar todas las referencias a la regla seleccionada.

Vista técnica

La pestaña Técnica muestra todas las clases y atributos para la regla que se está editando. Puede eliminar todas las clases y atributos pulsando de nuevo en el menú de contexto asociado con cada clase o atributo. Las acciones disponibles en la pestaña de negocio en la vista de diseño son las siguientes:

Vista esquema de clase: Esto muestra todas las reglas de nivel superior y las carpetas en las que están ubicadas. Hay un menú contextual disponible en cada elemento dentro de la vista esquema:

Tabla 13. Nuevos elementos de menú

Nombre	Descripción
Clase (Botón)	Crear una clase proporcionando un nombre de clase nuevo.
Atributo (Botón)	Crear un nuevo atributo dentro de una clase determinada proporcionando un nombre de atributo nuevo. El botón sólo estará habilitado una vez que se haya seleccionado una clase en la vista esquema, de lo contrario el botón permanece inactivo.
Nuevo atributo	Crear un nuevo atributo dentro de una clase determinada proporcionando un nombre de atributo nuevo.
Suprimir	Suprimir una clase o un atributo según lo que esté resaltado en la vista de árbol.
Establecer tipo de regla	Permite al desarrollador de reglas crear una derivación para un atributo.
Buscar referencias a esta regla	Permite al usuario buscar todas las referencias a la regla seleccionada.

Lienzo de diagrama

El lienzo de diagrama proporciona los controles para arrastrar y soltar, alternar técnico y enfocar y ampliar

Arrastrar y soltar

Un elemento de regla se puede arrastrar y soltar de cualquier paleta al diagrama. Por ejemplo, arrastrar y soltar un elemento "regla" en un atributo. Aparecerá un indicador visual en el destino si puede aceptar el elemento de regla que se está arrastrando.

Además se pueden arrastrar y soltar elementos de regla entre contenedores de elementos de regla. Por ejemplo, arrastre y suelte un elemento de "regla" de una sección de resultado del elemento de regla "cuando" en una sección de resultado de otro elemento de regla "cuando".

Alternar para mostrar diagramas detallados

El Editor CER proporciona dos tipos distintos de vistas para los elementos de regla. La vista de empresa es una versión simple destinada al escritor de reglas de empresa. La vista técnica es más detallada y es principalmente para el diseñador de reglas técnicas. El Editor CER proporciona un alternador para conmutar entre las vistas técnica y de empresa. El icono de conmutador se encuentra sobre los controles de enfoque y zoom en el lienzo de reglas.

Controles de enfoque y zoom

El editor CER proporciona controles de enfoque y zoom para permitir al usuario ver y editar mejor el conjunto de reglas. Las flechas en el control circular se pueden utilizar para enfocar las reglas de gran tamaño. Al pulsar en el botón central del control de enfoque se vuelve a centrar la imagen en la posición inicial. El botón más se puede utilizar para acercarse y el botón menos se puede utilizar para alejarse:

Herramientas y paletas de plantilla

El Editor CER proporciona cuatro tipos de paletas de elementos de regla y tres paletas de plantillas de regla. Estas paletas contienen lo siguientes Empresa (predeterminada), Empresa (ampliada), Tipos de datos y Técnica. Las plantillas se agrupan en lo siguiente: Unidades familiares, Unidades financieras, Unidades de ayuda alimenticia y la Tabla de decisiones.

Paletas empresariales

Las paletas empresariales, las predeterminadas y las ampliadas, incluyen un rango de elementos de regla que se pueden utilizar para diseñar la lógica empresarial en el formato de diagrama.

Tabla 14. Elementos de regla en paletas empresariales

Imagen	Nombre	Descripción
	Rule	El elemento Rule proporciona una representación gráfica de la expresión 'reference'. Consulte “Rule” en la página 115.
	Grupo de reglas And	El elemento Grupo de reglas And proporciona una representación de la expresión 'all' y devuelve un valor booleano. Consulte “Grupo de reglas And” en la página 117.

Tabla 14. Elementos de regla en paletas empresariales (continuación)

Imagen	Nombre	Descripción
	Grupo de reglas Or	El elemento Grupo de reglas Or proporciona una representación de la expresión 'any' y devuelve un valor booleano. Consulte “Grupo de reglas Or” en la página 118.
	Not	El elemento Not proporciona una representación gráfica de la expresión 'not' y niega un valor booleano. Consulte “Not” en la página 118.
	Choose	El elemento Choose proporciona una representación gráfica de la expresión 'choose' y elige un valor basándose en que se cumpla una condición. Consulte “Choose” en la página 118.
	Compare	El elemento Compare proporciona una representación gráfica de la expresión "compare" y compara un valor de la izquierda con un valor de la derecha, de acuerdo con la comparación proporcionada. Consulte “Comparar” en la página 119.
	When	El elemento When forma parte del elemento Elegir y contiene una condición a probar y un valor a devolver si la condición pasa. Consulte “When” en la página 119.
	Arithmetic	El elemento Arithmetic proporciona una representación gráfica de la expresión 'arithmetic' y realiza un cálculo aritmético de dos números (un número del lado izquierdo y un número del lado derecho) y opcionalmente redondea el resultado al número especificado de decimales. Consulte “Arithmetic” en la página 119.
	MIN	El elemento Min proporciona una representación gráfica de la expresión 'min' y determina el valor más pequeño en una lista (o nulo si la lista está vacía). Consulte “MIN” en la página 120.
	MAX	El elemento Max proporciona una representación gráfica de la expresión 'max' y determina el valor más grande en una lista (o nulo si la lista está vacía). Consulte “MAX” en la página 120.
	Sum	El elemento SUM proporciona una representación gráfica de la expresión 'sum' y calcula la suma numérica de una lista de valores de número. Consulte “SUM” en la página 120.
	Repeating Rule	El elemento Repeating Rule proporciona una representación gráfica de la expresión 'dynamiclist' y crea una nueva lista evaluando una expresión en cada elemento de una lista existente. Consulte “Repeating Rule” en la página 120.
	Filter	El elemento Filter proporciona una representación gráfica de la expresión 'filter' y crea una nueva lista que contiene todos los elementos de una lista existente que cumplen la condición de filtro. Consulte “Filter” en la página 121.

Tabla 14. Elementos de regla en paletas empresariales (continuación)

Imagen	Nombre	Descripción
	Size	El elemento Size proporciona una representación gráfica de la expresión 'property' y el nombre de la propiedad se establece en 'size'. Consulte “Size” en la página 122.
	Otherwise	El elemento Otherwise forma parte del elemento Elegir y contiene un valor que se debe devolver. Consulte “Otherwise” en la página 122.
	Legislation Change	El elemento Legislation Change proporciona una representación gráfica de 'legislationchange' y puede contener más de un elemento Era. Consulte “Legislation Change” en la página 122.
	Era	El elemento Era es una parte de los elementos de cambio de legislación y contiene una entrada de fecha (de vacío) y un valor para volver al elemento Cambio de legislación. Consulte “Era” en la página 123.

Paleta de tipos de datos

La paleta de tipos de datos incluye un rango de elementos de regla que se pueden utilizar para diseñar los tipos de datos en el formato de diagrama.

Tabla 15. Elementos de regla en la paleta de tipo de datos

Imagen	Nombre	Descripción
	Booleano	El elemento Booleano proporciona una representación gráfica de la expresión 'true' y 'false'. De forma predeterminada, el elemento Booleano está establecido en true. Consulte “Booleano” en la página 123.
	String	El elemento String proporciona una representación gráfica de la expresión 'String' que es un valor constante de número literal. Consulte “String” en la página 123.
	Number	El elemento Number proporciona una representación gráfica de la expresión 'Number' que es un valor constante de número literal. Consulte “Number” en la página 123.
	Date	El elemento Date proporciona una representación gráfica de la expresión 'Date' que es un valor constante de fecha literal. Consulte “Date” en la página 124.
	Codetable	El elemento Codetable proporciona una representación gráfica de la expresión 'Code' que es un valor constante literal que representa un código de una tabla de códigos de Cúram. Consulte “Codetable” en la página 124.
	Rate	El elemento Rate proporciona una representación gráfica de la expresión 'rate' que es un valor constante literal que representa una tasa de una tabla de tasas. Consulte “Rate” en la página 124.

Tabla 15. Elementos de regla en la paleta de tipo de datos (continuación)

Imagen	Nombre	Descripción
	Freqency Pattern	El elemento Freqency Pattern proporciona una representación gráfica de la expresión 'FrequencyPattern' que es un valor constante de FrequencyPattern literal. Consulte “Frequency Pattern” en la página 125.
	Resource Message	El elemento Resource Message de recurso proporciona una representación gráfica de la expresión 'ResourceMessage' y crea un mensaje traducible a partir de un recurso de propiedad. Consulte “Resource Message” en la página 125.
	XML message	El elemento XML message proporciona una representación gráfica de la expresión 'XMLMessage' y crea un mensaje traducible a partir de un recurso de propiedad. Consulte “XML Message” en la página 126.
	Null	El elemento Null proporciona una representación gráfica de 'null'. El elemento Null se puede utilizar en elementos como Choose, When, Compare, etc... para comparar cualquier valor con 'null'. Consulte “Nulo” en la página 126.

Paleta técnica

La paleta técnica incluye una serie de elementos de regla que se pueden utilizar para diseñar la lógica técnica en el formato de diagrama.

Tabla 16. Elementos de regla en la paleta técnica

Imagen	Nombre	Descripción
	Create	El elemento Create proporciona una representación gráfica de la expresión 'create' y obtiene una instancia nueva de una clase de regla en la memoria de la sesión. Consulte “Create” en la página 126.
	Search	El elemento Search proporciona una representación gráfica de la expresión 'readall' y recupera todas las instancias de objeto de regla de una clase de regla creadas por el código de cliente. Consulte “Search” en la página 127.
	Fixed List	El elemento Fixed list proporciona una representación gráfica de la expresión 'fixedlist' y crea una lista nueva de elementos conocidos en el tiempo de diseño de conjunto de reglas. Consulte “Fixed List” en la página 127.
	Property	El elemento Property proporciona una representación gráfica de la expresión 'property'. El elemento Property obtiene una propiedad de un objeto Java. Consulte “Property” en la página 128.
	Custom Expression	La expresión Custom proporciona una representación gráfica para cualquier nodo XML válido definido por el usuario. Consulte “Expresión personalizada” en la página 129.

Tabla 16. Elementos de regla en la paleta técnica (continuación)

Imagen	Nombre	Descripción
	Existence Timeline	El elemento Existence Timeline proporciona una representación gráfica de la expresión 'existencetimeline'. Consulte “Existence Timeline” en la página 129.
	Timeline	El elemento Timeline proporciona una representación gráfica de la expresión 'Timeline'. Consulte “Timeline” en la página 130.
	Interval	El elemento Interval proporciona una representación gráfica de la expresión 'Interval'. Consulte “Interval” en la página 130.
	Combine Succession Set	El elemento Combine Succession Set proporciona una representación gráfica de la expresión 'combineSuccessionSet'. Consulte “CombineSuccessionSet” en la página 131.
	Call	El elemento Call proporciona una representación gráfica de la expresión 'call' y llama a un método Java estático para realizar un cálculo complejo. Consulte “Call” en la página 131.
	Period Length	El elemento Period Length proporciona una representación gráfica de la expresión 'periodlength' y calcula la cantidad de unidades de tiempo entre dos fechas. Consulte “Period Length” en la página 132.
	ALL	El elemento ALL proporciona una representación gráfica de la expresión 'all' y devuelve un valor booleano. Consulte “ALL” en la página 132.
	ANY	El elemento ANY proporciona una representación gráfica de la expresión 'all' y devuelve un valor booleano. Consulte “ANY” en la página 133.
	This	El elemento This proporciona una representación gráfica de la expresión 'this' que es una referencia al objeto de regla actual. Consulte “This” en la página 133.
	Sort	El elemento Sort proporciona una representación gráfica de la expresión 'sort'. Clasificar toma una lista de elementos hijo y realiza la clasificación en ella. Consulte “Sort” en la página 133.
	Shared Rule Reference	La expresión Shared Rule Reference es básicamente una referencia normal que contiene un elemento Crear. Consulte “Referencia de regla compartida” en la página 133.
	CONCAT	El elemento Concat proporciona una representación gráfica de la expresión 'concat' y crea un mensaje concatenando una lista de valores. Consulte “CONCAT” en la página 134.
	Join Lists	El elemento JoinLists proporciona una representación gráfica de la expresión 'joinlists' y crea una lista nueva uniendo algunas listas existentes. Consulte “Unir listas” en la página 135.

Plantilla de unidades familiares

La Plantilla de unidades familiares incluye un rango de elementos de regla que se puede utilizar para diseñar las unidades familiares en formato de diagrama.

Tabla 17. Elementos de regla de la paleta de plantilla de unidades familiares

Imagen	Nombre	Descripción
	Composición de unidad familiar	Consulte "Composición de unidad familiar" en la página 135.
	Categoría de unidad familiar	Consulte "Categoría de unidad familiar" en la página 135.

Plantilla de unidades financieras

La plantilla de Unidades financieras incluye un rango de elementos de regla que se pueden utilizar para diseñar las unidades financieras en formato de diagrama.

Tabla 18. Elementos de regla de la paleta de plantilla de unidades financieras

Imagen	Nombre	Descripción
	Unidad financiera	Consulte "Unidad financiera" en la página 136.
	Categoría de unidad financiera	Consulte "Categoría de unidad financiera" en la página 136.
	Miembro de unidad financiera	Consulte "Miembro de unidad financiera" en la página 136.

Plantilla de unidades de ayuda alimenticia

La plantilla de Unidades de ayuda alimenticia incluye un rango de elementos de regla que se pueden utilizar para diseñar las unidades de ayuda alimenticia en formato de diagrama.

Tabla 19. Elementos de regla de la paleta de plantilla de unidades de ayuda alimenticia

Imagen	Nombre	Descripción
	Unidad de ayuda alimenticia	TAREA A REALIZAR: enlace a Ver unidad de ayuda alimenticia
	Categoría de persona única de ayuda alimenticia	TAREA A REALIZAR: enlace a Ver categoría de persona única de ayuda alimenticia "Categoría de persona única de ayuda alimenticia" en la página 136.
	Categoría multipersona de ayuda alimenticia	Consulte "Categoría multipersona de ayuda alimenticia" en la página 136.
	Miembros de grupo de comidas	Consulte "Miembros de grupo de comidas" en la página 137.
	Familiares	Consulte "Familiares" en la página 137.
	Descartar	Consulte "Descartar" en la página 137.
	Miembro de unidad familiar	Consulte "Miembro de unidad familiar" en la página 137.

Tabla 19. Elementos de regla de la paleta de plantilla de unidades de ayuda alimenticia (continuación)

Imagen	Nombre	Descripción
	Miembros opcionales	Consulte “Miembros opcionales” en la página 137.
	Excepciones	Consulte “Excepciones” en la página 137.

Plantilla de tabla de decisiones

La plantilla de tabla de decisiones contiene el elemento de tabla de decisiones que se utiliza para crear tablas de decisiones.

Tabla 20. Elementos de la paleta de la tabla de decisiones

Imagen	Nombre	Descripción
	Tabla de decisiones	La tabla de decisiones proporciona una representación gráfica de una 'tabla de decisiones'. Consulte “Decision Table” en la página 137 para obtener más información.

Menús emergentes de elemento de regla

El menú emergente del elemento de regla proporciona elementos generales (funciones) para todos los elementos de regla y elementos específicos (funciones) para un elemento de regla individual. Consulte “Referencia de elementos de regla para paletas de Editor CER” en la página 115 para conocer los elementos específicos del menú emergente del elemento de regla.

Tabla 21. Elementos de menús emergentes generales

Nombre	Descripción
Cortar	Cortar el elemento de regla y estar preparado para ir a otra ubicación.
Copiar	Copiar el elemento de regla existente y estar preparado para crear una nueva copia en otra ubicación.
Suprimir	Suprimir el elemento de regla del diagrama.
Contraer	Contraer los hijos del elemento de regla.
Expandir	Expandir los hijos del elemento de regla.
Crear línea de tiempo	Sólo está disponible para la vista técnica. Crear una línea de tiempo para el elemento de regla.
Crear intervalo de línea de tiempo	Sólo está disponible para la vista técnica. Crear un intervalo de línea de tiempo para el elemento de regla.
Eliminar línea de tiempo	Sólo está disponible para la vista técnica. Eliminar una línea de tiempo del elemento de regla.
Eliminar intervalo de línea de tiempo	Sólo está disponible para la vista técnica. Eliminar un intervalo de línea de tiempo del elemento de regla.

Propiedades de elemento de regla

En esta sección describe las propiedades generales, de clase de regla y de atributos. Consulte “Referencia de elementos de regla para paletas de Editor CER” en la página 115 para conocer las propiedades de elementos de regla específicos.

Paneles de propiedades y validación contraíbles

Los paneles de propiedades y validación se pueden ampliar y contraer pulsando el botón de conmutación en el borde de los contenedores de panel. Al ampliar los paneles de propiedades y validación se muestran los detalles de propiedad de diagrama seleccionados actualmente. Al contraer el panel de propiedades y validación, se ocultan estos detalles para proporcionar más espacio para ver el lienzo del diagrama.

Los paneles de propiedades se han agrupado por las pestañas de negocio y técnica para mostrar información de propiedad variada basándose en la perspectiva del desarrollador de reglas. Por ejemplo, no se espera que un desarrollador de reglas empresariales entre los códigos para un elemento de regla porque de este nivel de detalle se debe encargar el desarrollador de reglas técnicas.

Propiedades generales para todos los elementos de regla

Tabla 22. Propiedades generales

Nombre	Descripción
Nombre de visualización	Esto es un nombre traducible. El campo soporta caracteres de varios bytes y acentuados. Este campo está disponible en la pestaña Empresa.
Descripción	La descripción soporta entradas de texto libre para las reglas o los atributos. Los caracteres de varios bytes y acentuados están soportados. Es necesario que el desarrollador de reglas entre una descripción de negocio significativa de la regla o del atributo. Este campo está disponible en la pestaña Empresa. El contenido del campo de descripción es traducible y puede contener caracteres de varios bytes o acentuados.
Enlace de legislación	Enlace a cualquier sitio web de legislación. Este campo está disponible en la pestaña de negocio.
ID de nombre de visualización	ID para el nombre del elemento de regla. Este campo está disponible en la pestaña Técnica.
Etiqueta	La anotación de etiqueta soporta entradas de etiqueta de texto libre para cualquier elemento que soporte anotaciones. Se utiliza para buscar el elemento de regla. Este campo está disponible en la pestaña de negocio.

Propiedades de clase de regla

Tabla 23. Propiedades de clase de regla

Nombre	Descripción
Atributo primario	Seleccione un atributo en un menú desplegable para que sea un atributo principal de esta clase de regla. Esto puede verse en la pestaña Empresa.
Abstracto	La clase será una clase abstracta si se selecciona esta propiedad. Esto puede verse en la pestaña Empresa.
Ampliaciones	Los usuarios pueden ampliar esta clase de regla desde otra clase de regla que se pertenezca al conjunto de reglas actual o externo establecido por el asistente para cambiar el conjunto de reglas y la clase de regla. Esto puede verse en la pestaña Empresa.
Clase	Nombre de la clase de regla. Esto puede verse en la pestaña Técnica.

Tabla 23. Propiedades de clase de regla (continuación)

Nombre	Descripción
SuccessionSet	La anotación SuccessionSet se añadirá si se selecciona esta propiedad. Los usuarios deben seleccionar atributos de fecha en los desplegable "Atributo de fecha inicial" y "Atributo de fecha final". Esto puede verse en la pestaña Técnica.
ActiveInEditSuccessionSet	La anotación ActiveInEditSuccessionSet se añadirá si se selecciona esta propiedad. Los usuarios deben seleccionar atributos de fecha en los desplegable "Atributo de fecha inicial" y "Atributo de fecha final". Esto puede verse en la pestaña Técnica.

Propiedades del atributo

Tabla 24. Propiedades de atributo

Nombre	Descripción
Tipo de datos	El tipo de datos de un atributo. Si el usuario desea cambiar el tipo de datos a un línea de tiempo, seleccione el recuadro Línea de tiempo. Si el usuario desea cambiar el tipo de datos a una lista, seleccione el recuadro de lista. Esto puede verse en la pestaña Empresa.
Visualización	La anotación Visualización se añadirá si se selecciona esta propiedad. Requiere que los usuarios entren un valor. Esto puede verse en la pestaña Empresa.
Visualizar subpantalla	La anotación Visualizar subpantalla se añadirá si se selecciona esta propiedad. Esto puede verse en la pestaña Empresa.
Clase	Nombre de la clase de regla a la que pertenece el atributo. Esto puede verse en la pestaña Técnica.
Atributo	El nombre del atributo. Esto puede verse en la pestaña Técnica.
Conjunto de sucesiones relacionado	La anotación Conjunto de sucesiones relacionado se añadirá si se selecciona esta propiedad. Requiere que los usuarios seleccionen uno de ellos (ninguno, padre, hijo) en el menú desplegable. Esto puede verse en la pestaña Técnica.
Tipo de prueba relacionada	La anotación Tipo de prueba relacionada se añadirá si se selecciona esta propiedad. Requiere que los usuarios seleccionen uno de ellos (ninguno, padre, hijo) en el menú desplegable. Esto puede verse en la pestaña Técnica.
Abstracto	El atributo se establecerá en un elemento de regla "abstracto" y la clase será una clase abstracta. Esto puede verse en la pestaña Técnica.

Asistentes de elemento de regla

Los asistentes de clase de regla y cambio de conjunto de reglas son utilizados por algunos elementos de regla (por ejemplo, Regla, Crear elemento de regla) para enlazarse a otra clase de regla del conjunto de reglas actuales o bien del conjunto de reglas externas. Las tres opciones disponibles de este asistente son:

Tabla 25. Tabla de asistente de elemento de regla

Nombre	Descripción
Crear una referencia de regla vacía	Permite al escritor de reglas crear un marcador para un elemento de regla que puede editarse en un punto posterior mientras se desarrolla la regla.
Crear nueva regla	Permite al escritor de reglas crear una regla nueva especificando el nombre de la regla y el tipo de resultado de la nueva regla desde el recuadro combinado desplegable.
Utilizar regla existente	Permite al escritor de reglas elegir en una regla ya creada, seleccionando la regla en el recuadro combinado desplegable. Si la regla no está contenida en la regla existente, el escritor de reglas puede seleccionar otro conjunto de reglas especificando el nombre del conjunto de reglas y pulsando el botón de búsqueda. Se abrirá otro recuadro de diálogo permitiendo que el escritor de reglas elija el conjunto de reglas adecuado.

Referencia de elementos de regla para paletas de Editor CER

Las definiciones de todos los elementos de regla incluidas con el Editor CER. Los elementos de regla se categorizan por distintos tipos de paletas de elemento de regla; consulte la información anterior para saber categorizaciones útiles de estos elementos de regla.

Introducción

Esta sección define todos los elementos de regla incluidos con el Editor CER. Los elementos de regla siguientes se clasifican por diferentes tipos de paletas de elementos de regla; consulte las secciones anteriores para conocer clasificaciones útiles de estos elementos de regla.

Paletas

Las paletas pueden desacoplarse arrastrando la paleta de la posición original al lienzo del diagrama. Para volver a acoplar la paleta, simplemente vuelva a arrastrarla al lado derecho de la pantalla.

Paletas contraíbles

El contenedor de paleta puede expandirse y contraerse pulsando el botón de conmutador en el borde de los contenedores de la paleta. Al expandir el contenedor de la paleta se muestra el nombre de paleta seleccionado actualmente y las descripciones textuales de cada uno de los elementos de regla. Al contraer el contenedor de paleta se ocultan las descripciones textuales y se reduce la cantidad de espacio que el contenedor ocupa en el diagrama.

Referencia de elementos de regla para las paletas empresariales predeterminadas y ampliadas

Rule

El elemento Rule proporciona una representación gráfica de la expresión 'rule' y niega un valor booleano. No se pueden añadir otros elementos de paleta al elemento de referencia. El elemento Rule se puede añadir a otros elementos de paleta, por ejemplo Grupo de reglas And, Grupo de reglas Or o Repeating Rule. Existen siete tipos diferentes de escenarios para utilizar los elementos Rule. Una referencia puede añadirse a una regla cuando un usuario está en la vista Empresa

(consulte la sección 2.3.1 de *Trabajar con CER*) y en la vista Técnica (consulte la sección 2.3.2 de *Trabajar con CER*). Cuando un usuario está en la vista Empresa, están disponibles las opciones siguientes:

- Referencia vacía: los usuarios pueden crear una referencia vacía;
- Crear regla: los usuarios pueden crear una regla; y
- Utilizar regla existente: los usuarios pueden seleccionar una regla del conjunto de reglas actual o realizar una búsqueda de conjuntos de reglas externos y seleccionar una regla de un conjunto de reglas externo.

Cuando un usuario está en la vista Técnica, están disponibles las opciones siguientes:

- Utilizar regla existente: los usuarios pueden seleccionar una regla del conjunto de reglas actual o realizar una búsqueda de conjuntos de reglas externos y seleccionar una regla de un conjunto de reglas externo; y
- Crear regla: los usuarios pueden crear una regla.

Si un usuario está en la vista de empresa y desea añadir una referencia a un atributo o clase de regla determinados, dicho usuario deberá cambiar a la vista técnica.

Tabla 26. Tipos de escenarios para utilizar los elementos Regla

Nombre	Descripción
Referencia anidada	Se puede crear una referencia anidada en la referencia que apunta a un atributo que es de otro tipo de clase de regla, por ejemplo, no está contenido en la clase existente. El atributo de referencia más externo es un objeto de clase de atributo de referencia interna. Esta estructura puede crearse sólo si el atributo de referencia interna, que es de otro tipo de clase de regla, está ubicado en la clase donde se está creando la referencia interna.
Referencia anidada con Crear	Se puede crear una referencia anidada en la referencia que apunta a un atributo que es de otro tipo de clase de regla, por ejemplo, no está contenido en la clase existente, pero el atributo no está ubicado en la clase actual. El atributo de referencia más externo es un objeto de clase de atributo de referencia interna. Esta estructura sólo se puede crear si el atributo de referencia interna (que es de otro tipo de clase de regla) está ubicado en la clase que no es la clase donde se está creando la referencia interna.
CurrentUnitMemeber	Para hacer referencia al miembro de la unidad actual que satisface la condición de prueba excepcional.
FARelationship	Para hacer referencia a la clase que se utiliza como el registro de relación para el miembro del grupo de comidas.
FAException	Para hacer referencia a una clase que se utiliza para comprobar si otros miembros de una unidad satisfacen la condición excepcional.
HC Current	Para hacer referencia a un elemento de lista actual (como miembro de la unidad familiar, etc.) en la composición de unidad familiar, se utiliza HCCurrent.
Current	Para hacer referencia a un elemento de lista actual en las listas como dynamiclist, se utiliza el elemento Current. Si se tiene que hacer referencia a un atributo de la clase de elemento de lista actual, se reinicia alrededor del elemento actual una referencia que apunta a dicho atributo.

La tabla siguiente lista elementos de propiedades específicos para este elemento:

Tabla 27. Elementos de propiedades de referencia

Nombre	Descripción
Clase	Nombre de la clase de regla. Esto puede verse en la pestaña Empresa.
Atributo	Nombre de un atributo. Esto puede verse en la pestaña Empresa.
Elemento único	Sólo se devuelve un elemento del elemento. Esto puede verse en la pestaña Técnica.
Comportamiento cuando no se encuentran elementos	Devolver uno de estos resultados (error, devolver nulo) cuando no se encuentran elementos. Esto está activo cuando se marca el recuadro Elemento único.
Comportamiento cuando se encuentran varios elementos	Devolver uno de estos resultados (error, devolver nulo, devolver primero, devolver último) cuando se encuentran varios elementos. Esto está activo cuando se marca el recuadro Elemento único.

La tabla siguiente lista elementos de menú emergente específicos para este elemento.

Tabla 28. Elementos de menús emergentes de elemento Regla

Nombre	Descripción
Reiniciar en OR	Reiniciar el elemento de regla en el elemento de grupo de reglas Or.
Reiniciar en AND	Reiniciar el elemento de regla en el elemento de grupo de reglas And.
Editar regla	Edite el elemento de regla eligiendo la regla a la que desea hacer referencia. Consulte el asistente para editar regla de la manera siguiente.
Abrir diagrama para esta regla	Abre el diagrama para la regla a la que se está haciendo referencia en una nueva pestaña de diagrama.
Incluir lógica de reglas aquí	Si el elemento de regla está haciendo referencia a otra regla (en el mismo conjunto de reglas o en otro conjunto de reglas externo), la lógica para dicha regla se puede incluir en la regla actual que se está viendo en el editor.

Grupo de reglas And

El elemento Grupo de reglas And proporciona una representación de la expresión 'all' y devuelve un valor booleano. Opera en una lista de valores booleanos para determinar si todos los valores de lista son verdaderos. La lista de valores booleanos la proporciona normalmente una fixedlist. Otros elementos de la paleta, por ejemplo Referencia, Regla de repetición o Elegir, que proporcionan una lista de valores booleanos se pueden añadir al miembro vacío del elemento And para el cálculo. El elemento Grupo de reglas And se puede añadir a otros elementos de paleta, por ejemplo Grupos de reglas And, Grupos de reglas Or o Cuando.

La tabla siguiente lista elementos de menú emergente específicos para este elemento.

Tabla 29. Elementos de menús emergentes de elemento Grupo de reglas And

Nombre	Descripción
Reiniciar en AND	Reiniciar el elemento Grupo de reglas And en otro elemento Grupo de reglas And.
Reiniciar en OR	Reiniciar el elemento Grupo de reglas And en otro elemento Grupo de reglas Or.
Cambiar a O	Cambiar el elemento Grupo de reglas And por el elemento Grupo de reglas Or.

Grupo de reglas Or

El elemento Grupo de reglas Or proporciona una representación gráfica de los 'any' de los valores booleanos que se pueden añadir al miembro vacío del elemento Grupo de reglas Or para el cálculo. El elemento Grupo de reglas Or se puede añadir a otros elementos de paleta, por ejemplo Grupo de reglas And, Grupo de reglas Or o Cuando.

La tabla siguiente lista elementos de menú emergente específicos para este elemento.

Tabla 30. Elementos de menús emergentes de elemento Or

Nombre	Descripción
Reiniciar en grupo de reglas Or	Reiniciar el elemento Grupo de reglas Or en otro elemento Grupo de reglas Or.
Reiniciar en grupo de reglas And	Reiniciar el elemento Grupo de reglas Or en el elemento Grupo de reglas And.
Cambiar a And	Cambiar el elemento Grupo de reglas Or por el elemento Grupo de reglas And.

Not

El elemento Not proporciona una representación gráfica de la expresión 'not' y niega un valor booleano. Otros elementos de paleta, por ejemplo Referencia, Or y And, que devuelven un valor booleano se pueden añadir al elemento Not. El elemento Not se puede añadir a otros elementos de paleta, por ejemplo Grupo de reglas And, Grupo de reglas Or o Regla de repetición.

La tabla siguiente lista elementos de menú emergente específicos para este elemento.

Tabla 31. Elementos de menús emergentes de elemento Not

Nombre	Descripción
Reiniciar en And	Reiniciar el elemento Not en otro elemento Grupo de reglas And.
Reiniciar en Or	Reiniciar el elemento Not en el elemento Grupo de reglas Or.

Choose

El elemento Choose proporciona una representación gráfica de la expresión 'choose' y elige un valor basándose en que se cumpla una condición. El asistente para editar elección proporciona nueve tipos de datos que son serie, número, booleano, fecha, fecha y hora, tabla de códigos, mensaje, línea de tiempo, tabla de códigos, clase de regla, clase Java y mensaje. Sólo se pueden añadir los elementos Cuando y

Otherwise al elemento Elegir. El elemento Choose se puede añadir a otros elementos de paleta, por ejemplo Grupo de reglas And, Grupo de reglas Or o la vista compleja de la comparación.

La tabla siguiente lista elementos de menú emergente específicos para este elemento.

Tabla 32. Elementos de menús emergentes de elemento Elegir

Nombre	Descripción
Reiniciar en OR	Reiniciar el elemento Choose en el elemento Grupo de reglas Or.
Reiniciar en AND	Reiniciar el elemento Choose en el elemento Grupo de reglas And.
Editar elección	Proporcionar una lista de tipos de datos para Choose. Consulte el recuadro de diálogo Editar elección a continuación.

Comparar

El elemento Compare proporciona una representación gráfica de la expresión "compare" y compara un valor de la izquierda con un valor de la derecha, de acuerdo con la comparación proporcionada. Cuando una comparación se añade a un diagrama, crea miembros vacíos para los argumentos del lado izquierdo y del lado derecho. El elemento Compare se puede añadir a otros elementos de paleta, por ejemplo When, Grupo de reglas And o Repeating Rule.

La tabla siguiente lista elementos de menú emergente específicos para este elemento.

Tabla 33. Elementos de menús emergentes de elemento Compare

Nombre	Descripción
Reiniciar en OR	Reiniciar el elemento Compare en el elemento Grupo de reglas Or.
Reiniciar en AND	Reiniciar el elemento Compare en el elemento de grupo de reglas And.

When

El elemento When forma parte del elemento Choose y contiene una condición a probar y un valor a devolver si la condición pasa. Se pueden añadir otros elementos de paleta, por ejemplo Code o Any, a la condición vacía del elemento When. Se pueden añadir otros elementos de paleta, por ejemplo Number o Reference, al valor vacío del elemento When.

Arithmetic

El elemento Arithmetic proporciona una representación gráfica de la expresión 'arithmetic' y realiza un cálculo aritmético de dos números (un número del lado izquierdo y un número del lado derecho) y opcionalmente redondea el resultado al número especificado de decimales. Otros elementos de la paleta, por ejemplo Máx, Mín o Referencia se pueden añadir al elemento Aritmetic. El elemento Arithmetic se puede añadir a otros elementos de paleta, por ejemplo When o Repeating Rule

La tabla siguiente lista elementos de propiedades específicos para este elemento:

Tabla 34. Elementos de propiedades de Aritmética

Nombre	Descripción
Posiciones decimales	Un lugar para que un usuario entre el número de decimales. Esto puede verse en la pestaña Empresa.
Redondeo	Proporcione diferentes tipos de redondeo (tope, abajo, mínimo, mitad hacia abajo, mitad igual, mitad hacia arriba, arriba). Esto puede verse en la pestaña Empresa.

MIN

El elemento Min proporciona una representación gráfica de la expresión 'min' y determina el valor más pequeño en una lista (o nulo si la lista está vacía). El elemento Min tiene una fixedlist que contiene cualquier tipo de objeto comparable. Otros elementos de paleta, por ejemplo número o referencia de regla, pueden añadirse al miembro vacío (fixedlist) del elemento Min. El elemento Min se puede añadir a otros elementos de paleta, por ejemplo Cuando o Regla de repetición.

La tabla siguiente lista elementos de menú emergente específicos para este elemento.

Tabla 35. Elementos de menús emergentes de elemento Min

Nombre	Descripción
Cambiar a MAX	Cambiar el elemento MIN por el elemento MAX.

MAX

El elemento Max proporciona una representación gráfica de la expresión 'max' y determina el valor más grande en una lista (o nulo si la lista está vacía). El elemento Max tiene una fixedlist que contiene cualquier tipo de objeto comparable. Otros elementos de paleta, por ejemplo el número o la referencia de regla, pueden añadirse al miembro vacío (fixedlist) del elemento Max. El elemento Max puede añadirse a otros elementos de paleta, por ejemplo Cuando o Regla de repetición.

La tabla siguiente lista elementos de menú emergente específicos para este elemento.

Tabla 36. Elementos de menús emergentes de elemento Max

Nombre	Descripción
Cambiar a MIN	Cambiar el elemento Max por el elemento MIN.

SUM

El elemento SUM proporciona una representación gráfica de la expresión 'sum' y calcula la suma numérica de una lista de valores de número. El elemento SUM tiene una fixedlist que contiene cualquier tipo de objeto numérico. Otros elementos de paleta, por ejemplo el número o la referencia de regla pueden añadirse al miembro vacío (fixedlist) del elemento SUM. El elemento SUM se puede añadir a otros elementos de paleta, por ejemplo Cuando o Regla de repetición.

Repeating Rule

El elemento Repeating Rule proporciona una representación gráfica de la expresión 'dynamiclist' y crea una nueva lista evaluando una expresión en cada elemento de una lista existente. Se pueden añadir otros elementos de paleta, por ejemplo Referencia de regla, a la lista vacía del elemento Repeating Rule. Se pueden añadir otros elementos de paleta, por ejemplo Suma o Elegir al miembro vacío

(listitemexpression) del elemento Repeating Rule. El elemento Repeating Rule se puede añadir a otros elementos de paleta, por ejemplo Grupo de reglas And, Grupo de reglas Or o Cuando.

La tabla siguiente lista elementos de propiedades específicos para este elemento:

Tabla 37. Elementos de propiedades de regla de repetición

Nombre	Descripción
Elemento único	Sólo se devuelve un elemento del elemento. Esto puede verse en la pestaña Técnica.
Comportamiento cuando no se encuentran elementos	Devolver uno de estos resultados (error, devolver nulo) cuando no se encuentran elementos. Esto está activo cuando se marca 'Elemento único'.
Comportamiento cuando no se encuentran elementos	Devolver uno de estos resultados (error, devolver nulo) cuando no se encuentran elementos. Esto está activo cuando se marca 'Elemento único'.

La tabla siguiente lista elementos de menú emergente específicos para este elemento.

Tabla 38. Elementos de menús emergentes de regla de repetición

Nombre	Descripción
Eliminar duplicados	Eliminar elementos duplicados en el elemento de regla de repetición.
Concatenar resultados	Concatenar los elementos en el elemento de regla de repetición.
Unir listas internas	Unir las listas para crear una sola lista.
Éxito en cualquiera	Reiniciar el elemento de regla de repetición en Any.
Éxito en todos	Reiniciar el elemento de regla de repetición en All.
Sumar elementos	Sumar una lista de números.

Filter

El elemento Filter proporciona una representación gráfica de la expresión 'filter' y crea una nueva lista que contiene todos los elementos de una lista existente que cumplen la condición de filtro. Se pueden añadir otros elementos de paleta, por ejemplo Referencia, a la lista vacía del elemento de filtro. Se pueden añadir otros elementos de paleta, por ejemplo Suma o Elegir, al miembro vacío (listitemexpression) del elemento de filtro. El elemento Filter puede añadirse a otros elementos de paleta, por ejemplo And, Or o Cuando.

La tabla siguiente lista elementos de propiedades específicos para este elemento:

Tabla 39. Elementos de propiedades de filtro

Nombre	Descripción
Elemento único	Sólo se devuelve un elemento del elemento. Esto puede verse en la pestaña Técnica.
Comportamiento cuando no se encuentran elementos	Devolver uno de estos resultados (error, devolver nulo) cuando no se encuentran elementos. Esto está activo cuando se marca 'Elemento único'.

Tabla 39. Elementos de propiedades de filtro (continuación)

Nombre	Descripción
Comportamiento cuando se encuentran varios elementos	Devolver uno de estos resultados (error, devolver nulo, devolver primero, devolver último) cuando se encuentran varios elementos. Esto está activo cuando se marca 'Elemento único'.

La tabla siguiente lista elementos de menú emergente específicos para este elemento.

Tabla 40. Elementos de menús emergentes de Filtro

Nombre	Descripción
Eliminar duplicados	Eliminar elementos duplicados en el elemento de filtro.
Concatenar resultados	Concatenar los elementos en el elemento de filtro.
Unir listas internas	Unir las listas para crear una sola lista.
Reiniciar en OR	Reiniciar el elemento de filtro en el elemento de grupo de reglas Or.
Reiniciar en AND	Reiniciar el elemento de filtro en el elemento de grupo de reglas And.

Size

El elemento Size proporciona una representación gráfica de la expresión 'property' y el nombre de este elemento se establece en 'size'. El elemento Size obtiene una propiedad de un objeto Java. Otros elementos de paleta, por ejemplo Rule Reference o Repeating Rule, se pueden añadir al elemento Size. El elemento Size puede añadirse a otros elementos de paleta, por ejemplo When o Repeating Rule.

La tabla siguiente lista elementos de propiedades específicos para este elemento:

Tabla 41. SElementos de propiedades de Size

Nombre	Descripción
Valor	Nombre del elemento de propiedad.

Otherwise

El elemento Otherwise forma parte del elemento Elegir y contiene un valor que se debe devolver. Se pueden añadir otros elementos de paleta, por ejemplo Número o Referencia de regla, al valor vacío del elemento Otherwise.

Legislation Change

El elemento Legislation Change proporciona una representación gráfica de 'legislationchange' y puede contener más de un elemento Era. El asistente de cambio de legislación proporciona diez tipos de datos (serie, número, booleano, fecha, fecha y hora, lista, mensaje, tabla de códigos, clase de regla y clase Java). El tipo de datos seleccionado se utiliza para definir un valor de devolución del elemento Época.

La tabla siguiente lista elementos de menú emergente específicos para este elemento.

Tabla 42. Elementos de menús emergentes de elemento Cambio de legislación

Nombre	Descripción
Editar cambio de legislación	Editar el cambio de legislación eligiendo un tipo para la nueva regla. Consulte el recuadro de diálogo de Editar cambio de legislación a continuación.

Era

El elemento Era es una parte de los elementos de cambio de legislación y contiene una entrada de fecha (de vacío) y un valor para volver al elemento Cambio de legislación. Otros elementos de la paleta, por ejemplo la referencia de regla o fecha pueden añadirse a la entrada de fecha vacía del elemento de época. Se pueden añadir otros elementos de paleta, por ejemplo Choose o Reference, al valor vacío del elemento de época.

Referencia de elementos de regla para paleta de tipos de datos

Booleano

El elemento Booleano proporciona una representación gráfica de la expresión 'true' y 'false'. De forma predeterminada, el elemento Booleano está establecido en true. El elemento Booleano se puede añadir a otros elementos de paleta, por ejemplo los elementos Grupo de reglas And, Grupo de reglas Or o Regla de repetición. No se puede añadir ningún elemento al elemento booleano

La tabla siguiente lista elementos de menú emergente específicos para este elemento.

Tabla 43. Elementos de menús emergentes de Booleano

Nombre	Descripción
Cambiar a False	Cambiar el elemento de regla booleano a falso (false) si su valor es verdadero (true).
Cambiar a True	Cambiar el elemento de regla booleano a verdadero (true) si su valor es falso (false).

String

El elemento String proporciona una representación gráfica de la expresión 'String' que es un valor constante de número literal. El elemento String se puede añadir a otros elementos de paleta, por ejemplo When o Repeating Rule. No se puede añadir ningún elemento al elemento String.

La tabla siguiente lista elementos de propiedades específicos para este elemento:

Tabla 44. Elementos de propiedades de String

Nombre	Descripción
Valor	El valor de la serie. Esto puede verse en la pestaña Empresa.

Number

El elemento Number proporciona una representación gráfica de la expresión 'Number' que es un valor constante de número literal. El elemento Number se puede añadir a otros elementos de paleta, por ejemplo Cuando o Regla de repetición. No se puede añadir ningún elemento al elemento de número.

La tabla siguiente lista elementos de propiedades específicos para este elemento:

Tabla 45. Elementos de propiedades de número

Nombre	Descripción
Valor	El valor del número. Esto puede verse en la pestaña Empresa.

Date

El elemento Date proporciona una representación gráfica de la expresión 'Date' que es un valor constante de fecha literal. El elemento Date se puede añadir a otros elementos de paleta, por ejemplo PeriodLength o Era. No se puede añadir ningún elemento al elemento Fecha.

La tabla siguiente lista elementos de propiedades específicos para este elemento:

Tabla 46. Elementos de propiedades de Date

Nombre	Descripción
Valor	El valor de la fecha. El valor predeterminado se establece a la fecha de hoy. Esto puede verse en la pestaña Empresa.
Fecha cero	Al marcar la casilla de verificación 'Fecha cero', el usuario puede utilizar la 'Fecha cero' de Cúram. Esto puede verse en la pestaña Empresa.

Codetable

El elemento Codetable proporciona una representación gráfica de la expresión 'Code' que es un valor constante literal que representa un código de una tabla de códigos de Cúram. El elemento CodeTable se puede añadir a otros elementos de paleta, por ejemplo Cuando. No se puede añadir ningún elemento al elemento CodeTable.

La tabla siguiente lista elementos de propiedades específicos para este elemento:

Tabla 47. Elementos de propiedades de CodeTable

Nombre	Descripción
Nombre de tabla de códigos	Nombre de la tabla de códigos. Déjelo en blanco y busque todas las tablas de códigos disponibles. Especifique un valor y busque una tabla de códigos que empiece por el valor especificado. Esto puede verse en la pestaña Empresa.
Valor de tabla de códigos	Un recuadro desplegable que contiene todos los elementos contenidos en la tabla de códigos seleccionada. Esto puede verse en la pestaña Empresa.

Rate

El elemento Rate proporciona una representación gráfica de la expresión 'rate' que es un valor constante literal que representa una tasa de una tabla de tasas de Cúram. El elemento Rate se puede añadir a otros elementos de paleta, por ejemplo Cuando. No se puede añadir ningún elemento al elemento Rate.

La tabla siguiente lista elementos de propiedades específicos para este elemento:

Tabla 48. Elementos de propiedades de Tasa

Nombre	Descripción
Nombre de tabla	Nombre de la tabla de tasas. Esto puede verse en la pestaña Empresa.
Fila	El valor de la fila de la tabla de tasas. Esto puede verse en la pestaña Empresa.
Columna	El valor de columna de la tabla de tasas. Esto puede verse en la pestaña Empresa.

Frequency Pattern

El elemento Frequency Pattern de frecuencia proporciona una representación gráfica de la expresión 'FrequencyPattern' que es un valor constante de FrequencyPattern literal. El elemento Frequency Pattern se puede añadir a otros elementos de paleta, por ejemplo Cuando o Crear. No se puede añadir ningún elemento al elemento Frequency Pattern.

La tabla siguiente lista elementos de propiedades específicos para este elemento:

Tabla 49. FElementos de propiedades de Frequency Pattern

Nombre	Descripción
Patrón	El patrón de frecuencia. Esto puede verse en la pestaña Empresa.

La tabla siguiente lista elementos de menú emergente específicos para este elemento.

Tabla 50. Elementos de menús emergentes de Filtro

Nombre	Descripción
Editar patrón de frecuencia	Editar el patrón de frecuencia seleccionado.

Resource Message

El elemento Resource Message proporciona una representación gráfica de la expresión 'ResourceMessage' y crea un mensaje traducible a partir de un recurso de propiedad. El elemento Resource Message se puede añadir a otros elementos de paleta, por ejemplo Cuando o Crear. No se puede añadir ningún elemento al elemento Resource Message.

La tabla siguiente lista elementos de propiedades específicos para este elemento:

Tabla 51. Elementos de propiedades de Resource Message

Nombre	Descripción
Clave	Los objetos de paquete de recursos contienen una matriz de pares de clave-valor. Especifique la clave, que debe ser una serie, cuando desee recuperar el valor del paquete de recursos. Esto puede verse en la pestaña Empresa.
Paquete de recursos	Nombre del paquete de recursos. Esto puede verse en la pestaña Empresa.

La tabla siguiente lista elementos de menú emergente específicos para este elemento.

Tabla 52. Elementos de menús emergentes de mensaje de recursos

Nombre	Descripción
Nuevo argumento	Añadir un argumento nuevo al elemento de regla de mensaje de recurso.
Eliminar argumento	Eliminar un argumento del elemento de regla de mensaje de recurso.

XML Message

El elemento XML Message proporciona una representación gráfica de la expresión 'XmlMessage' y crea un mensaje a partir del contenido XML de formato libre. El elemento XML Message se puede añadir a otros elementos de paleta, por ejemplo When o Create. No se puede añadir ningún elemento al elemento XML Message.

La tabla siguiente lista elementos de propiedades específicos para este elemento:

Tabla 53. Elementos de propiedades de XML Message

Nombre	Descripción
Valor	El valor de contenido del recurso Xml.

La tabla siguiente lista elementos de menú emergente específicos para este elemento.

Tabla 54. Elementos de menús emergentes de XML Message

Nombre	Descripción
Editar mensaje	Proporciona un recuadro de diálogo en el que un usuario puede entrar el contenido del mensaje.

Nulo

El elemento nulo (null) proporciona una representación gráfica de los 'null'. El elemento Null puede utilizarse en elementos como Elegir/Cuando, Comparar, etc. para comparar cualquier valor con nulo.

Referencia de elementos de regla para paleta de Lógica técnica

Create

El elemento Create proporciona una representación gráfica de la expresión 'create' y obtiene una instancia nueva de una clase de regla en la memoria de la sesión. El elemento Create soporta dos tipos de parámetros (estándares y obligatorios). Otros elementos de paleta, por ejemplo Rule Reference, Repeating Rule o Choose, se pueden añadir a los parámetros del elemento Create. El elemento Create se puede añadir a otros elementos de paleta, por ejemplo Repeating Rule o When

La tabla siguiente lista elementos de propiedades específicos para este elemento:

Tabla 55. Crear elementos de propiedades

Nombre	Descripción
Clase	Nombre de la clase de regla que se elige como un tipo. Esto puede verse en la pestaña Técnica.

La tabla siguiente lista elementos de menú emergente específicos para este elemento.

Tabla 56. Elementos de menús emergentes de elemento Crear

Nombre	Descripción
Editar Create	Editar el elemento Create eligiendo la clase de regla y el conjunto de reglas a los que desea hacer referencia.
Parámetro nuevo	Cree un parámetro nuevo seleccionando el atributo para el que desea añadir un parámetro.
Nuevo parámetro obligatorio	Crear un nuevo parámetro obligatorio.

Search

El elemento Search proporciona una representación gráfica de la expresión 'readall' y recupera todas las instancias de objeto de regla de una clase de regla creadas por el código de cliente. Puede recuperar un único elemento de una lista si se selecciona la expresión 'singleitem'. El elemento Search se puede añadir a otros elementos de paleta, por ejemplo Filtro o Crear. No se puede añadir ningún elemento al elemento Search.

La tabla siguiente lista elementos de propiedades específicos para este elemento:

Tabla 57. Elementos de propiedades de Buscar

Nombre	Descripción
Clase	Nombre de la clase de regla que se elige como un tipo. Esto puede verse en la pestaña Técnica.
Conjunto de reglas	Nombre del conjunto de reglas que incluye la clase de regla elegida. Esto puede verse en la pestaña Técnica.
Elemento único	Sólo se devuelve un elemento del elemento. Esto puede verse en la pestaña Técnica.
Comportamiento cuando no se encuentran elementos	Devolver uno de estos resultados (error, devolver nulo) cuando no se encuentran elementos. Esto se activa cuando se marca el recuadro 'Elemento único'.
Comportamiento cuando se encuentran varios elementos	Devolver uno de estos resultados (error, devolver nulo, devolver primero, devolver último) cuando se encuentran varios elementos. Esto se activa cuando se marca el recuadro 'Elemento único'.

La tabla siguiente lista elementos de menú emergente específicos para este elemento.

Tabla 58. Elementos de menús emergentes de elemento Buscar

Nombre	Descripción
DEditar Search	Editar el elemento Search eligiendo la clase de regla y el conjunto de reglas a los que desea hacer referencia.

Fixed List

El elemento Fixed List proporciona una representación gráfica de la expresión 'fixedlist' y crea una lista nueva de elementos conocidos en el tiempo de diseño de conjunto de reglas. El asistente de lista fija proporciona nueve tipos de datos que son serie, número, booleano, fecha, fecha y hora, entrada de tabla de códigos, clase de regla, clase Java y mensaje. Se proporciona la función de sublista. Otros

elementos de paleta, por ejemplo Referencia de regla, Regla de repetición o Elegir, se pueden añadir al miembro vacío del elemento Fixed List. El elemento Fixed List se puede añadir a otros elementos de paleta, por ejemplo Grupo de reglas And, Grupo de reglas Or o Cuando. El elemento Fixed List se envolverá dentro de otro elemento de regla dependiendo del tipo de datos que se haya seleccionado. Por ejemplo, los elementos de regla Grupo de reglas And/Grupo de reglas Or para el tipo booleano, el elemento de regla Concatenación para el tipo de serie, los elementos de regla Máx/Mín/Suma para el tipo de número.

La tabla siguiente lista elementos de propiedades específicos para este elemento:

Tabla 59. Elementos de propiedades de FixedList

Nombre	Descripción
Tipo de datos	El tipo de datos de la FixedList. Debería coincidir con el tipo de datos del atributo. Si el usuario desea cambiar el tipo de datos a un línea de tiempo, seleccione el recuadro Línea de tiempo. Si el usuario desea cambiar el tipo de datos a una lista, seleccione el recuadro de lista. Esto puede verse en la pestaña Empresa.

Property

El elemento Property proporciona una representación gráfica de la expresión 'property'. El elemento Property obtiene una propiedad de un objeto Java. Otros elementos de paleta, por ejemplo, Referencia de regla o Regla de repetición se pueden añadir al elemento Property. El elemento Property se puede añadir a otros elementos de paleta, por ejemplo Cuando o Regla de repetición.

La tabla siguiente lista elementos de propiedades específicos para este elemento:

Tabla 60. Elementos de propiedades de Propiedad

Nombre	Descripción
Valor	Nombre del elemento de propiedad.

La tabla siguiente lista elementos de menú emergente específicos para este elemento.

Tabla 61. Elementos de menús emergentes de Propiedad

Nombre	Descripción
Reiniciar en OR	Reiniciar el elemento de propiedad en el elemento de grupo de reglas Or.
Reiniciar en AND	Reiniciar el elemento de propiedad en el elemento de grupo de reglas And.
Nuevo argumento	Añadir un argumento nuevo al elemento de regla de propiedad.
Eliminar argumento	Eliminar un argumento del elemento de regla de propiedad.

PRECAUCIÓN:
Desde Cúram V6, CER y el Gestor de dependencias soportan el recálculo automático de valores calculados por CER si cambian sus dependencias.

Si cambia la implementación de un método de propiedad, CER y el Gestor de dependencias *no* sabrán recalcular automáticamente los valores de atributo que se han calculado utilizando la versión antigua del método de propiedad.

Una vez que se ha utilizado un método de propiedad en un entorno de producción para valores de atributo almacenados, en lugar de cambiar la implementación deberá crear un nuevo método de propiedad (con la nueva implementación necesaria) y cambiar los conjuntos de reglas para utilizar el nuevo método de propiedad. Cuando publique los cambios de conjunto de reglas para que apunten al nuevo método de propiedad, CER y el Gestor de dependencias recalcularán automáticamente todas las instancias de los valores de atributo afectados.

Expresión personalizada

La expresión personalizada proporciona una representación gráfica para cualquier nodo XML válido definido por el usuario. La expresión personalizada se puede añadir a cualquier elemento en CER y es obligatoria para que el usuario se asegure de que los nombres de nodo de expresión personalizada no coinciden con los nombres de nodo de lenguaje CER.

La tabla siguiente lista elementos de menú emergente específicos para este elemento.

Tabla 62. Elementos de menús emergentes de expresión personalizada

Nombre	Descripción
Editar expresión personalizada	Visualizar la ventana emergente para editar la expresión personalizada.

Existence Timeline

El elemento Existence Timeline proporciona una representación gráfica de la expresión 'existencetimeline'. Otros elementos de la paleta, por ejemplo la fecha o la referencia de regla, pueden añadirse a FromDate y ToDate del elemento de línea de tiempo de existencia. Otros elementos de paleta, por ejemplo la fecha o la referencia de regla, se pueden añadir a preExistenceValue, postExistenceValue y ExistenceValue del elemento de línea de tiempo de existencia. El elemento Existence Timeline se puede añadir a otros elementos de paleta, por ejemplo Repeating Rule.

La tabla siguiente lista elementos de propiedades específicos para este elemento:

Tabla 63. Elementos de propiedades de línea de tiempo de existencia

Nombre	Descripción
Tipo de datos	El tipo de datos de la línea de tiempo de existencia. Si el usuario desea cambiar el tipo de datos a un línea de tiempo, seleccione el recuadro Línea de tiempo. Si el usuario desea cambiar el tipo de datos a una lista, seleccione el recuadro de lista. Esto puede verse en la pestaña Empresa.

La tabla siguiente lista elementos de menú emergente específicos para este elemento.

Tabla 64. Elementos de menús emergentes de elemento de línea de tiempo de existencia

Nombre	Descripción
Editar línea de tiempo de existencia	Proporcionar 10 tipos de datos (serie, número, booleano, fecha, fecha y hora, entrada de tabla de códigos, clase de regla, clase Java, lista y mensaje) para un tipo de intervalo del elemento de regla de línea de tiempo de existencia.

Timeline

El elemento Timeline proporciona una representación gráfica de la expresión 'Timeline'. Un valor inicial se puede establecer para un elemento Timeline utilizando la opción de menú en él. Otros elementos de paleta, por ejemplo Interval, FixedList, Repeating Rule, etc., se pueden añadir al elemento Timeline.

La tabla siguiente lista elementos de propiedades específicos para este elemento:

Tabla 65. Elementos de propiedades de línea de tiempo de existencia

Nombre	Descripción
Tipo de datos	El tipo de datos de la línea de tiempo. Si el usuario desea cambiar el tipo de datos a un línea de tiempo, seleccione el recuadro Línea de tiempo. Si el usuario desea cambiar el tipo de datos a una lista, seleccione el recuadro de lista. Esto puede verse en la pestaña Empresa.

La tabla siguiente lista elementos de menú emergente específicos para este elemento.

Tabla 66. Elementos de menús emergentes de elemento Timeline

Nombre	Descripción
Añadir intervalos	Añade intervalos a la línea de tiempo
Añadir valor inicial	Añade el valor inicial a la línea de tiempo
Eliminar intervalos	Elimina intervalos de la línea de tiempo
Eliminar valor inicial	Elimina el valor inicial de la línea de tiempo
Editar línea de tiempo	Muestra el asistente de Línea de tiempo para editar la línea de tiempo

Interval

El elemento Interval proporciona una representación gráfica de la expresión 'Interval'. Otros elementos de paleta, por ejemplo fecha o referencia, pueden añadirse a FromDate y ToDate del elemento de línea de tiempo de existencia. El intervalo sólo se puede añadir al elemento Interval de un elemento de línea de tiempo. El tipo de intervalo se puede establecer utilizando el asistente de intervalo. El elemento Interval contiene un elemento inicial y un elemento de valor como elementos hijo.

La tabla siguiente lista elementos de menú emergente específicos para este elemento.

Tabla 67. Elementos de menús emergentes de elemento Intervalo

Nombre	Descripción
Editar intervalo	Muestra el asistente para editar el intervalo

CombineSuccessionSet

El elemento CombineSuccessionSet proporciona una representación gráfica de la expresión 'combineSuccessionSet'. Otros elementos de paleta, por ejemplo Filtro o Lista fija, se pueden añadir al elemento CombineSuccessionSet. El elemento CombineSuccessionSet se puede añadir a otros elementos de paleta, por ejemplo Cuando o Crear.

La tabla siguiente lista elementos de menú emergente específicos para este elemento.

Tabla 68. Elementos de menús emergentes de elemento CombineSuccessionSet

Nombre	Descripción
Editar CombineSuccessionSet	Editar el elemento CombineSuccessionSet eligiendo la clase de regla y el conjunto de reglas a los que desea hacer referencia.

Call

El elemento Call proporciona una representación gráfica de la expresión 'call' y llama a un método Java estático para realizar un cálculo complejo. El argumento nuevo puede añadirse al elemento de llamada. Se pueden añadir otros elementos de paleta, por ejemplo la fecha, la duración del periodo o la referencia de regla al argumento del elemento de llamada. El elemento Period Length se puede añadir a otros elementos de paleta, por ejemplo Create.

La tabla siguiente lista elementos de propiedades específicos para este elemento:

Tabla 69. Elementos de propiedades de Call

Nombre	Descripción
Clase	El nombre de la clase que tiene el método de llamada. Esto puede verse en la pestaña Técnica.
Nombre de método	El nombre del método al que se llamará. Esto puede verse en la pestaña Técnica.
Tipo de datos	El tipo de devolución de la llamada. Debería coincidir con el tipo de datos del atributo. Si el usuario desea cambiar el tipo de datos a un línea de tiempo, seleccione el recuadro Línea de tiempo. Si el usuario desea cambiar el tipo de datos a una lista, seleccione el recuadro de lista. Esto puede verse en la pestaña Empresa.

La tabla siguiente lista elementos de menú emergente específicos para este elemento.

Tabla 70. Elementos de menús emergentes de Llamar

Nombre	Descripción
Reiniciar en AND	Reiniciar el elemento de regla de llamada en el elemento de grupo de reglas And.
Reiniciar en OR	Reiniciar el elemento de regla de llamada en el elemento de grupo de reglas Or.
Nuevo argumento	Añadir un argumento nuevo al elemento de regla de llamada.
Eliminar argumento	Eliminar un argumento del elemento de regla de llamada.

PRECAUCIÓN:
Desde Cúram V6, CER y el Gestor de dependencias soportan el recálculo automático de valores calculados por CER si cambian sus dependencias.

Si cambia la implementación de un método estático, CER y el Gestor de dependencias *no* sabrán recalcular automáticamente los valores de atributo que se han calculado utilizando la versión anterior del método estático.

Una vez que se ha utilizado un método estático en un entorno de producción para los valores de atributo almacenados, en lugar de cambiar la implementación deberá crear un método estático nuevo (con la nueva implementación necesaria) y cambiar los conjuntos de reglas para utilizar el método estático nuevo. Cuando publique los cambios de conjunto de reglas para que apunten al nuevo método estático, CER y el Gestor de dependencias recalcularán automáticamente todas las instancias de los valores de atributo afectados.

Period Length

El elemento Period Length proporciona una representación gráfica de la expresión 'periodlength' y calcula la cantidad de unidades de tiempo entre dos fechas. Se pueden añadir otros elementos de paleta, por ejemplo la fecha, la llamada o la referencia de regla, al elemento de duración de periodo. El elemento Period Length se puede añadir a otros elementos de paleta, por ejemplo Crear.

La tabla siguiente lista elementos de propiedades específicos para este elemento:

Tabla 71. Elementos de propiedades de Duración del periodo

Nombre	Descripción
Unidad	Proporcionar cuatro tipos de unidades (días, semanas, meses, años) para el elemento de regla de duración de periodo. Esto puede verse en la pestaña Técnica.
EndDateInclusion	Proporcionar dos tipos de inclusiones de fecha (inclusivo o exclusivo). Esto puede verse en la pestaña Técnica.

ALL

El elemento ALL proporciona una representación gráfica de la expresión 'all' y devuelve un valor booleano. Opera en una lista de valores booleanos para determinar si todos los valores de lista son verdaderos. El elemento ALL no tiene ninguna fixedlist. Otros elementos de la paleta, por ejemplo Regla de repetición o Lista fija, se pueden añadir al elemento ALL. El elemento ALL se puede añadir a otros elementos de paleta, por ejemplo Regla de repetición.

La tabla siguiente lista elementos de menú emergente específicos para este elemento.

Tabla 72. Elementos de menús emergentes de elemento ALL

Nombre	Descripción
Reiniciar en OR	Reiniciar el elemento Grupo de reglas And en otro elemento Grupo de reglas Or.
Reiniciar en AND	Reiniciar el elemento Grupo de reglas And en otro elemento Grupo de reglas And.
Cambiar a O	Cambiar el elemento Grupo de reglas And por el elemento Grupo de reglas Or.

ANY

El elemento ANY proporciona una representación gráfica de la expresión 'any' y devuelve un valor booleano. Opera en una lista de valores booleanos para determinar si alguno de los valores de lista es verdadero. El elemento ANY no tiene ninguna fixedlist. Otros elementos de paleta, por ejemplo Regla de repetición o Lista fija, se pueden añadir al elemento ANY. El elemento ANY se puede añadir a otros elementos de paleta, por ejemplo Regla de repetición.

La tabla siguiente lista elementos de menú emergente específicos para este elemento.

Tabla 73. Elementos de menús emergentes de elemento ANY

Nombre	Descripción
Reiniciar en OR	Reiniciar el elemento Grupo de reglas Or en otro elemento Grupo de reglas Or.
Reiniciar en AND	Reiniciar el elemento Grupo de reglas Or en el elemento Grupo de reglas And.
Cambiar a AND	Cambiar el elemento Grupo de reglas Or por el elemento Grupo de reglas And.

This

El elemento This proporciona una representación gráfica de la expresión 'this' que es una referencia al objeto de regla actual. El elemento This se puede añadir a otros elementos de paleta, por ejemplo When o Any. No se puede añadir ningún elemento al elemento This.

Sort

El elementoSort proporciona una representación gráfica de la expresión 'sort'. Sort toma una lista de elementos hijo y realiza la clasificación en ella.

La tabla siguiente lista elementos de menú emergente específicos para este elemento.

Tabla 74. Elementos de menús emergentes de elemento ANY

Nombre	Descripción
Nuevos elementos de clasificación ascendente	Permite añadir un elemento de clasificación ascendente.
Nuevos elementos de clasificación descendente	Permite añadir un elemento de clasificación descendente.

Referencia de regla compartida

La referencia de regla compartida es básicamente una referencia normal que contiene un elemento Crear. La referencia de regla compartida muestra un asistente que visualiza los nombres de las clases de regla que tienen el atributo Primary establecido en el conjunto de reglas actual. La referencia de regla compartida se puede editar seleccionando la opción de menú "Editar referencia de regla compartida" en el diagrama. Una regla compartida es una clase con un atributo primario en ella. El asistente de regla compartida permite al usuario crear reglas compartidas que se pueden utilizar para crear las referencias de regla compartida.

La tabla siguiente lista elementos de propiedades específicos para este elemento:

Tabla 75. Elementos de propiedades de referencia de regla compartida

Nombre	Descripción
Clase	Nombre de la clase de regla. Esto puede verse en la pestaña Empresa.
Atributo	Nombre de un atributo. Esto puede verse en la pestaña Empresa.
Elemento único	Sólo se devuelve un elemento del elemento. Esto puede verse en la pestaña Técnica.
Comportamiento cuando no se encuentran elementos	Devolver uno de estos resultados (error, devolver nulo) cuando no se encuentran elementos. Esto está activo cuando se marca el recuadro Elemento único.
Comportamiento cuando se encuentran varios elementos	Devolver uno de estos resultados (error, devolver nulo, devolver primero, devolver último) cuando se encuentran varios elementos. Esto está activo cuando se marca el recuadro Elemento único.

La tabla siguiente lista elementos de menú emergente específicos para este elemento.

Tabla 76. Elementos de menús emergentes de elemento Referencia de regla compartida

Nombre	Descripción
Reiniciar en OR	Reiniciar el elemento de referencia de regla compartida en el elemento de grupo de regla Or.
Reiniciar en AND	Reiniciar el elemento de referencia de regla compartida en el elemento de grupo de regla And.
Editar referencia	Editar el elemento de referencia eligiendo la regla a la que desea hacer referencia.
Editar referencia de regla compartida	Editar el elemento de referencia de regla compartida eligiendo la regla compartida a la que desea hacer referencia. Consulte el recuadro de diálogo de referencia de edición de regla compartida a continuación.
Parámetro nuevo	Cree un parámetro nuevo seleccionando el atributo para el que desea añadir un parámetro. Consulte el recuadro de diálogo Parámetro nuevo a continuación.
Nuevo parámetro obligatorio	Crear un nuevo parámetro obligatorio.

CONCAT

El elemento Concatenar (Concat) proporciona una representación gráfica de la expresión 'concat' y crea un mensaje localizable concatenando una lista de valores. El elemento Concatenar tiene una fixedlist que contiene objetos de serie. Otros elementos de paleta, por ejemplo la serie o la referencia de regla pueden añadirse al miembro vacío (fixedlist) del elemento Concatenar. El elemento Concatenar puede añadirse a otros elementos de paleta, por ejemplo Cuando o Regla de repetición.

La tabla siguiente lista elementos de propiedades específicos para este elemento:

Tabla 77. Elementos de propiedades de Concatenar

Nombre	Descripción
Tipo de datos	El tipo de datos del elemento Concatenar. Debe coincidir con el tipo de datos del atributo. Si el usuario desea cambiar el tipo de datos a un línea de tiempo, seleccione el recuadro Línea de tiempo. Si el usuario desea cambiar el tipo de datos a una lista, seleccione el recuadro de lista. Esto puede verse en la pestaña Empresa.

Unir listas

El elemento JoinLists proporciona una representación gráfica de la expresión 'joinlists' y crea una lista nueva uniendo algunas listas existentes. Se pueden añadir otros elementos de paleta, por ejemplo Referencia de regla o Elegir, al miembro vacío (fixedlist) del elemento JoinLists. El elemento JoinLists se puede añadir a otros elementos de paleta, por ejemplo Grupo de reglas And, Grupo de reglas Or o Cuando.

La tabla siguiente lista elementos de propiedades específicos para este elemento:

Tabla 78. Elementos de propiedades de Unir listas

Nombre	Descripción
Elemento único	Sólo se devuelve un elemento del elemento.
Comportamiento cuando no se encuentran elementos	Devolver uno de estos resultados (error, devolver nulo) cuando no se encuentran elementos.
Comportamiento cuando se encuentran varios elementos	Devolver uno de estos resultados (error, devolver nulo, devolver primero, devolver último) cuando se encuentran varios elementos.

La tabla siguiente lista elementos de menú emergente específicos para este elemento.

Tabla 79. Elementos de menús emergentes de Unir listas

Nombre	Descripción
Eliminar duplicados	Eliminar elementos duplicados en el elemento Unir listas.
Concatenar resultados	Concatenar los elementos en el elemento Unir listas.
Unir listas internas	Unir las listas para crear una sola lista.

Referencia de elementos de reglas para plantillas de unidades familiares

Composición de unidad familiar

Esta plantilla de composición se utiliza en las reglas de CGIS (Curam Global Income Support) y presenta una composición de unidad familiar.

Categoría de unidad familiar

Esta plantilla representa una nueva categoría de unidad familiar para una unidad familiar dentro de las reglas de CGIS.

La tabla siguiente lista elementos de menú emergente específicos para este elemento.

Tabla 80. Elementos de menús emergentes de Categoría de unidad familiar

Nombre	Descripción
Añadir miembro obligatorio	Añadir un nuevo miembro obligatorio al elemento de regla de categoría de unidad familiar.
Eliminar miembro obligatorio	Eliminar un miembro obligatorio del elemento de regla de categoría de unidad familiar.
Añadir miembro opcional	Añadir un nuevo miembro opcional al elemento de regla de categoría de unidad familiar.
Eliminar miembro opcional	Eliminar un miembro opcional del elemento de regla de categoría de unidad familiar.

Referencia de elementos de regla para plantillas de unidades financieras

Unidad financiera

Esta plantilla se utiliza en las reglas de Soporte de ingresos global de Curam y presenta una unidad financiera.

Categoría de unidad financiera

Esta plantilla representa una nueva categoría de unidad financiera para una unidad financiera de las reglas de CGIS.

Miembro de unidad financiera

Esta plantilla representa un nuevo miembro de unidad financiera en las reglas de CGIS.

Referencia de elementos de regla para plantillas de unidades de Ayuda alimenticia

Unidad de ayuda alimenticia

Esta plantilla representa una nueva unidad de ayuda alimenticia en las reglas de CGIS.

Un diagrama de ayuda alimenticia debe configurarse primero de poderse editar por completo. Algunos asistentes del editor requieren que se establezca la configuración antes de proporcionar opciones de menú específicas de ayuda alimenticia, por ejemplo el asistente de referencia. Una configuración de diagrama de ayuda alimenticia se puede cambiar en cualquier momento.

Categoría de persona única de ayuda alimenticia

Esta plantilla representa una nueva categoría de persona única de ayuda alimenticia en las reglas de CGIS.

Categoría multipersona de ayuda alimenticia

Esta plantilla representa una nueva categoría multipersona de ayuda alimenticia en las reglas de CGIS.

La tabla siguiente lista elementos de menú emergente específicos para este elemento.

Tabla 81. Elementos de menús emergentes de categoría multipersona de ayuda alimenticia

Nombre	Descripción
Eliminar grupo de comidas	Eliminar un grupo de comida del elemento de regla Categoría multipersona de ayuda alimenticia.
Eliminar familiares	Eliminar familiares del elemento de regla Categoría multipersona de ayuda alimenticia.
Eliminar Descartar	Eliminar Descartar del elemento de regla Categoría multipersona de ayuda alimenticia.
Eliminar miembros cabeza de unidad familiar	Eliminar miembros cabeza de unidad familiar del elemento de regla Categoría multipersona de ayuda alimenticia.
Eliminar miembro opcional	Eliminar un miembro opcional del elemento de regla Categoría multipersona de ayuda alimenticia.

Miembros de grupo de comidas
Esta plantilla representa nuevos miembros de grupo de comidas de ayuda alimenticia en las reglas de CGIS.

Familiares
Esta plantilla representa nuevos familiares de ayuda alimenticia en las reglas de CGIS.

Descartar
Esta plantilla representa el nuevo descarte de ayuda alimenticia en las reglas de CGIS.

Miembro de unidad familiar
Esta plantilla representa un nuevo miembro de ayuda alimenticia de la unidad familiar en las reglas de CGIS.

Miembros opcionales
Esta plantilla representa miembros opcionales nuevos de ayuda alimenticia en las reglas de CGIS.

Excepciones
Esta plantilla representa nuevas excepciones de ayuda alimenticia en las reglas de CGIS.

Referencia de elementos de regla para plantillas de tabla de decisiones

Decision Table
El elemento Decision Table proporciona una representación gráfica de la expresión 'tabla de decisiones'. Cuando se arrastra una tabla de decisiones a una regla o un atributo, se muestra el asistente **Crear tabla de decisiones**. El usuario debe establecer las opciones siguientes:

- Número de filas: es el número de filas que estarán contenidas en la tabla de decisiones, con un número máximo de filas de 99.
- Tipo de resultado: es el tipo de retorno de la tabla de decisiones. El tipo de resultado debe coincidir con el tipo de resultado de la regla o atributo que contiene la tabla de decisiones; y
- Clase de regla: los usuarios pueden seleccionar la clase de regla actual, o cambiar clases de reglas.

Cuando el usuario pulse el botón *Siguiente*, podrá utilizar una regla existente o bien optar por crear una regla.

La tabla siguiente lista elementos de propiedades específicos para este elemento:

Tabla 82. Elementos de propiedades de Tabla de decisiones

Nombre	Descripción
Clase	Nombre de la clase de regla que se elige como un tipo. Esto puede verse en la pestaña Empresa.
Atributo	El nombre del atributo que contienen la tabla de decisiones. Esto puede verse en la pestaña Empresa.
Elemento único	Sólo se devuelve un elemento del elemento. Esto puede verse en la pestaña Técnica.
Comportamiento cuando no se encuentran elementos	Devolver uno de estos resultados (error, devolver nulo) cuando no se encuentran elementos. Esto se activa cuando se marca el recuadro 'Elemento único'.
Comportamiento cuando se encuentran varios elementos	Devolver uno de estos resultados (error, devolver nulo, devolver primero, devolver último) cuando se encuentran varios elementos. Esto se activa cuando se marca el recuadro 'Elemento único'.

La tabla siguiente lista elementos de menú emergente específicos para este elemento.

Tabla 83. Elementos de menús emergentes de la tabla de decisiones

Nombre	Descripción
Editar tabla de decisiones	Cambiar el tipo de resultado o el atributo asociado.
Añadir fila nueva.	Añade una nueva fila a la tabla de decisiones.

Procedimientos recomendados de CER

Siga estas prácticas recomendadas al escribir conjuntos de reglas CER para que sea más fácil desarrollar, probar y mantener los conjuntos de reglas.

El atributo de regla `description`

Cada clase de regla hereda básicamente de una “clase de regla raíz” de CER. Esta clase raíz incluye un atributo de regla `description` que tiene una implementación predeterminada (pero no especialmente útil).

El valor de un atributo `description` de objeto de regla se produce en RuleDoc y también mediante el método `toString` en un `RuleObject` (que muchos IDE Java utilizando cuando se “pulsa” en una variable). Como tal, una descripción (`description`) significativa puede ser indispensable al comprender el comportamiento del conjunto de reglas.

Debe alterar temporalmente el cálculo de `description` predeterminado creando explícitamente un atributo `description` en cada una de las clases de regla. Puede utilizar el Editor CER para crear un atributo `description` como se crea un atributo normal en cualquier clase de regla. El validador de conjunto de reglas CER emitirá un aviso si tiene una clase de regla que no define (o hereda de otra clase de regla definida) un atributo de regla `description`.

El atributo`description` es un mensaje traducible y (como otros atributos de regla) su cálculo puede ser tan simple o tan complejo como sea necesario.

A continuación se muestra un conjunto de reglas de ejemplo, donde algunas clases de regla proporcionan una implementación de `description`:

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_description"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="firstName">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="lastName">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- Alterar temporalmente la descripción predeterminada -->
    <Attribute name="description">
      <type>
        <javaclass name="curam.creole.value.Message"/>
      </type>
      <derivation>
        <!-- Concatenar el nombre y apellido de la persona -->
        <concat>
          <fixedlist>
            <listof>
              <javaclass name="Object"/>
            </listof>
            <members>

              <reference attribute="firstName"/>
              <String value=" "/>
              <reference attribute="lastName"/>
            </members>
          </fixedlist>
        </concat>
      </derivation>
    </Attribute>

  </Class>

  <Class name="Income">
    <!-- La persona con la que este
         registro de ingresos está relacionado. -->
    <Attribute name="person">
      <type>
        <ruleclass name="Person"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>
```

```
   <Attribute name="startDate">
     <type>
       <javaclass name="curam.util.type.Date"/>
     </type>
     <derivation>
       <specified/>
     </derivation>
   </Attribute>
   <Attribute name="amount">
     <type>
       <javaclass name="Number"/>
     </type>
     <derivation>
       <specified/>
     </derivation>
   </Attribute>

   <!-- Alterar temporalmente la descripción predeterminada -->
   <Attribute name="description">
     <type>
       <javaclass name="curam.creole.value.Message"/>
     </type>
     <derivation>
       <!-- Concatenar la descripción de la persona y la fecha
de inicio-->
       <concat>
         <fixedlist>
           <listof>
             <javaclass name="Object"/>
           </listof>
           <members>

             <reference attribute="description">
               <reference attribute="person"/>
             </reference>
             <!-- En un conjunto de reglas real, esta descripción utilizará
                  un <ResourceMessage> para evitar series codificadas
                  monolingües.  -->
             <String value="'s income, starting on "/>
             <reference attribute="startDate"/>
           </members>
         </fixedlist>
       </concat>
     </derivation>
   </Attribute>

 </Class>

 <Class name="Benefit">
   <!-- NB ninguna alteración temporal de <description>; el validador de conjunto de reglas CER
        emitirá un aviso y los objetos de regla esta clase
        serán más difíciles de entender en RuleDoc o un
        entorno de desarrollo integrado. -->
   <!-- La persona con la que este
        registro de prestación está relacionado. -->
   <Attribute name="person">
     <type>
       <ruleclass name="Person"/>
     </type>
     <derivation>
       <specified/>
     </derivation>
   </Attribute>

   <Attribute name="amount">
     <type>
       <javaclass name="Number"/>
```

```
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>
  </Class>

</RuleSet>
```

Y aquí se muestra una clase de prueba que crea algunos objetos de regla (`Person`, `Income` y `Benefit`):

```
package curam.creole.example;

import java.io.File;

import junit.framework.TestCase;
import curam.creole.execution.session.RecalculationsProhibited;
import curam.creole.execution.session.Session;
import curam.creole.execution.session.SessionDoc;
import curam.creole.execution.session.Session_Factory;
import
 curam.creole.execution.session.StronglyTypedRuleObjectFactory;
import curam.creole.ruleclass.Example_description.impl.Benefit;
import
 curam.creole.ruleclass.Example_description.impl.Benefit_Factory;
import curam.creole.ruleclass.Example_description.impl.Income;
import
 curam.creole.ruleclass.Example_description.impl.Income_Factory;
import curam.creole.ruleclass.Example_description.impl.Person;
import
 curam.creole.ruleclass.Example_description.impl.Person_Factory;
import curam.creole.storage.inmemory.InMemoryDataStorage;
import curam.util.type.Date;

/**
 * Prueba el atributo de regla de descripción.
 */
public class TestDescription extends TestCase {

  /**
   * Prueba el atributo de regla de descripción.
   */
  public void testDescriptions() {

    /**
     * Crear una sesión nueva.
     */
    final Session session =
        Session_Factory.getFactory().newInstance(
            new RecalculationsProhibited(),
            new InMemoryDataStorage(
                new StronglyTypedRuleObjectFactory()));

    /**
     * Crear un SessionDoc para informar sobre objetos de regla.
     */
    final SessionDoc sessionDoc = new SessionDoc(session);

    /*
     * Crear un objeto de regla Person.
     */
    final Person person =
        Person_Factory.getFactory().newInstance(session);
    person.firstName().specifyValue("Juan");
    person.lastName().specifyValue("Herrero");
```

```
    /*
     * Crear un objeto de regla Income.
     */
    final Income income =
        Income_Factory.getFactory().newInstance(session);
    income.person().specifyValue(person);
    income.amount().specifyValue(123);
    income.startDate().specifyValue(Date.fromISO8601("20070101"));

    /*
     * Crear un objeto de regla Benefit.
     */
    final Benefit benefit =
        Benefit_Factory.getFactory().newInstance(session);
    benefit.person().specifyValue(person);
    benefit.amount().specifyValue(234);

    /*
     * El método .toString evalúa el atributo de regla
     * de descripción
     */
    System.out.println(person.toString());

    /*
     * println llama a un método toString de un objeto para imprimirlo.
     */
    System.out.println(income);

    /*
     * La clase de regla de prestación no proporciona una implementación
     * del atributo de regla de descripción, por lo tanto aquí obtendremos una
     * descripción predeterminada
     */
    System.out.println(benefit);

    /*
     * Escribir SessionDoc para esta sesión.
     */
    sessionDoc.write(new File("./gen/sessiondoc"));
>
  }

}
```

Cuando la prueba se ejecuta, produce esta salida, mostrando las descripciones de objeto de regla:

```
Juan Herrero
Ingresos de Juan Herrero, a partir del 01/01/07 00:00
Instancia no descrita de la clase de regla 'Benefit', id '3'
```

Al final de la prueba, los objetos de regla de la sesión se producen como SessionDoc. El resumen de SessionDoc de alto nivel muestra los objetos de regla creados y lista la descripción de cada objeto de regla:

Example_description

Generated: 13-Jul-2012 11:52:43

External rule objects

Details	Type	Description	Action
details	Example_description.Benefit	Undescribed instance of rule class 'Benefit', id '3'	Created during this session
details	Example_description.Income	John Smith's income, starting on 01/01/07 00:00	Created during this session
details	Example_description.Person	John Smith	Created during this session

Internal rule objects

Details	Type	Description	Action

Figura 16. SessionDoc que muestra los valores de `description` de objeto de regla

La descripción para el objeto de regla `Benefit` es la descripción predeterminada; en ausencia de una buena implementación de `description`, alguien que lea el SessionDoc puede tener que navegar al SessionDoc para el objeto de regla `Benefit` a fin de comprenderlo:

Rule Object

Generated: 13-Jul-2012 11:52:43

Type

Example_description.Benefit

Description

Undescribed instance of rule class 'Benefit', id '3'

Creation

Created externally

Action during this session

Created during this session

Attributes

Name	Declared type	State	Value	Derivation	Depends on	Used by
amount	Number	SPECIFIED	234	• Specified externally.	*None*	*None*
description	Message	CALCULATED	Undescribed instance of rule class 'Benefit', id '3'	• Default rule object description.	*None*	*None*
person	Person	SPECIFIED	John Smith	• Specified externally.	*None*	*None*

Figura 17. SessionDoc para un objeto de regla sin alteración temporal de `description`

Por último, esta captura de pantalla muestra cómo un entorno de desarrollo integrado (por ejemplo Eclipse, mostrado) utiliza el método `toString` de un objeto al depurar, que (para los objetos de regla) calcula la descripción (`description`):

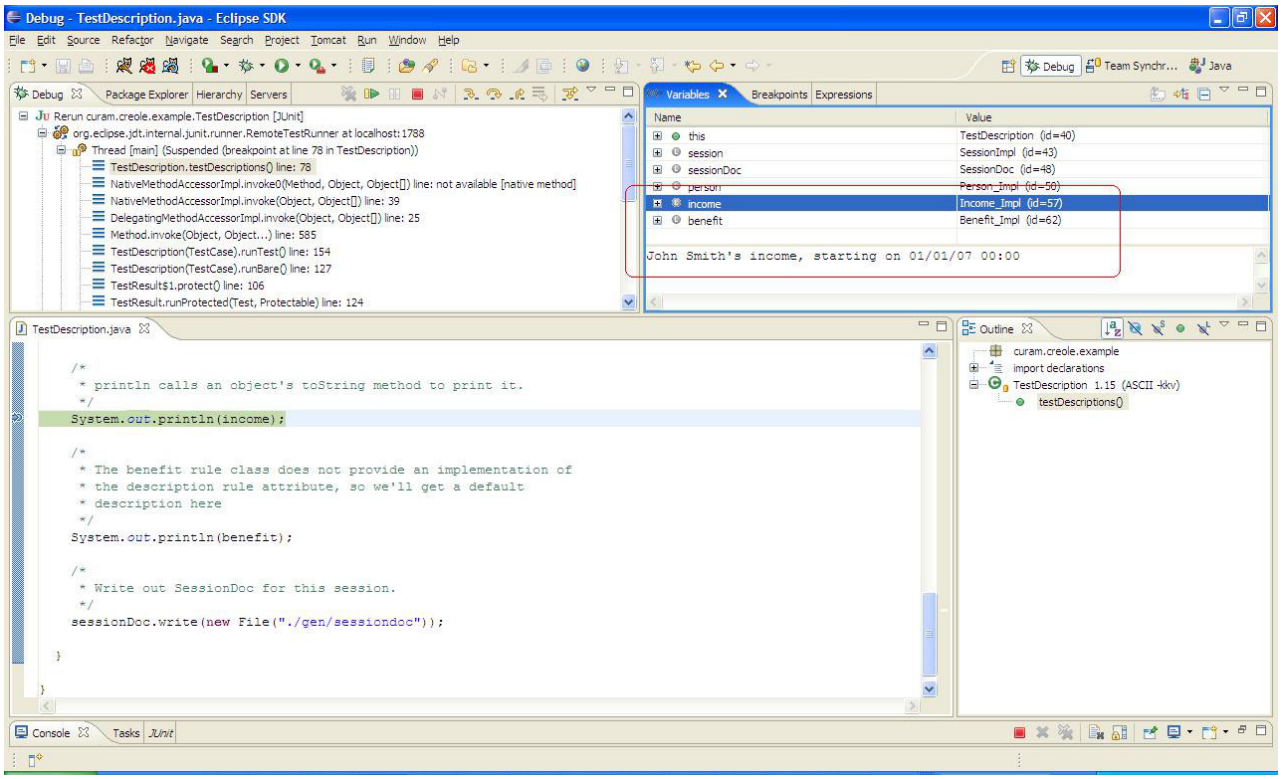

Figura 18. Uso de `description` en un entorno de desarrollo integrado

Consejo: Recuerde que la finalidad de `description` es describir una instancia de objeto de regla, no la clase de regla propiamente dicha.

En concreto, el cálculo del atributo de regla `description` debe incluir datos, lo que significa que es fácil distinguir entre instancias de objeto de regla diferentes de la clase de regla.

Obtención de un conjunto de reglas que funciona rápidamente

Puede ser útil tener un conjunto de reglas CER rápidamente "activo y en ejecución" aplazando determinadas tareas de desarrollo de reglas.

Considere la posibilidad de utilizar uno o varios de los siguientes atajos:

- Crear clases de regla vacías (es decir ningún atributo de regla) para cada uno de los conceptos empresariales; se pueden añadir atributos de regla más tarde. Por ejemplo, puede utilizar el Editor CER para crear una clase de regla vacía y añadir un atributo a ella más adelante.
- Crear derivaciones codificadas para atributos de regla; se pueden añadir reglas empresariales más tarde. Por ejemplo, si se declara que el cálculo de un atributo de regla `isEligible sea <true>`, es posible que pueda grabar pruebas y/o integrar CER con su propia aplicación: puede sustituir la derivación "siempre elegible" codificada por las normas empresariales reales más adelante. Por ejemplo, puede utilizar el Editor CER para arrastrar el elemento de regla "Booleano" (su valor predeterminado está establecido en true) a un atributo de regla `isEligible`.

- Crear mensajes como series codificadas simples de un solo entorno local; puede convertir las series en mensajes traducibles más tarde. Por ejemplo, puede utilizar el Editor CER para arrastrar el elemento de regla "Mensaje de recurso" o "Mensaje XML" a un atributo de regla.
- Utilizar valores de serie en lugar de valores de tabla de código de aplicación; puede convertir las series a códigos posteriormente (y actualizar las reglas que los prueban). Por ejemplo, puede utilizar el Editor CER para arrastrar el elemento de regla "Serie" a un atributo de regla.

Importante: El hecho de simplemente tomar estos atajos *no* elimina la necesidad de realizar el trabajo más rigurosamente, sólo aplaza parte de este trabajo hasta después del punto en el que el conjunto de reglas "ejecutable".

No debe postergar la creación de pruebas para las reglas; en particular, si toma estos atajos, una gran parte de las pruebas de conjunto de reglas ayudará a evitar que se entren determinados tipos de error cuando se deshacen los atajos. Cree las pruebas a medida que escribe las reglas.

También puede estar tentado a aplazar la creación de atributos de regla `description` para las clases de regla; sin embargo, en las primeras etapas del diseño de conjunto de reglas, un atajo de este tipo puede resultar ser economía falsa, porque los atributos de regla `description` pueden ayudarle mucho en la depuración de reglas, a un coste relativamente bajo.

Elementos de regla de denominación

Los atributos de regla CER deben denominarse de forma que describan su significado empresarial. Denomine un atributo de regla según el resultado que proporciona, en lugar de describir cómo lo proporciona el atributo de regla.

Un nombre de conjunto de reglas puede contener una combinación de letras (no acentuadas, en el rango de caracteres A-Z estándar), números y caracteres de subrayado. Un nombre de conjunto de reglas debe empezar por una letra en mayúsculas.

Un nombre de clase puede contener una combinación de letras (no acentuadas, en el rango de caracteres A-Z estándar), números y caracteres de subrayado. Un nombre de clase debe empezar por una letra en mayúsculas.

Un nombre de atributo puede contener una combinación de letras (no acentuadas, en el rango de caracteres A-Z estándar), números y caracteres de subrayado. Un nombre de atributo debe empezar por una letra minúscula.

Consejo: Utilice "camelCase" para denominar los elementos de regla.

camelCase es la práctica de escritura de palabras compuestas o frases en las que los elementos se unen sin espacios, con la letra inicial de cada elemento en mayúsculas dentro del compuesto y la primera letra en mayúscula o minúscula.

Nota: Cada elemento de regla tiene un nombre único que no se puede localizar. Para obtener descripciones localizables, utilice la anotación de "Etiqueta", como se describe en “Localización de descripciones de artefacto de regla CER” en la página 11.

Cuándo utilizar la expresión de referencia

El uso correcto de la expresión `reference` es clave para la estructura de un conjunto de reglas CER correcto. El uso de `reference` va de la mano con la creación del número correcto de atributos de regla. El Editor CER proporciona distintos tipos de escenarios para crear y utilizar el elemento de referencia de regla. Consulte el elemento "regla" en “Rule” en la página 115.

Encontrar el equilibrio correcto (entre pocos usos de `reference` y demasiados) es quizá más arte que ciencia; sin embargo, a continuación se proporcionan algunas directrices generales:

- Si encuentra que algunas de las expresiones están muy profundamente anidadas o son complejas, es posible que tenga demasiado pocas referencias. Considere la posibilidad de dividir expresiones complejas creando atributos de regla para un bloque significativo de expresiones y utilizando en su lugar una referencia (`reference`) al atributo de regla nuevo.
- Si los requisitos de tener un fuerte concepto o cálculo que no se correlaciona claramente con un atributo de regla, debe considerar la posibilidad de crear un atributo de regla.
- Si varias expresiones repiten la misma clase de cálculos, el conjunto de reglas puede beneficiarse de la creación de un atributo de regla para implementar la lógica común.
- Si encuentra que un atributo de regla es difícil de denominar, el atributo de regla podría ser una encapsulación innecesaria de lógica y es posible que tenga demasiados usos de `reference` en el conjunto de reglas. Considere la posibilidad de eliminar el atributo de regla y de "incorporar" la derivación en los lugares donde se utiliza, especialmente en el caso donde sólo se hace referencia al atributo de regla desde otro cálculo.

Utilizar RuleDoc

Cuando los conjuntos de reglas CER tienen un tamaño pequeño, puede abrirlos directamente en el Editor CER y entenderlos por completo fácilmente.

Sin embargo, a medida que aumenta la complejidad del conjunto de reglas, puede conocer mejor la estructura y el comportamiento del conjunto de reglas aprovechando la herramienta RuleDoc de CER.

Consulte “RuleDoc” en la página 19 para saber cómo generar RuleDoc desde los conjuntos de reglas CER.

Normalizar reglas comunes

A medida que desarrolla el conjunto de reglas, puede que observe reglas que son similares en diferentes partes de la funcionalidad de conjunto de reglas.

Deberá tratar de identificar las reglas comunes y centralizarlas.

En términos generales tiene dos opciones disponibles al centralizar reglas comunes:

- **Herencia**

  Utilice el soporte de CER para la herencia de implementación a fin de permitir que una clase de regla amplíe otra. El Editor CER proporciona el mecanismo de herencia para diseñar la regla. Consulte el elemento "ampliaciones" en “Propiedades de clase de regla” en la página 113.
- **Contención**

Utilice el soporte de CER para crear nuevos objetos de regla a partir de reglas a fin de permitir que una clase de regla cree nuevas instancias de otra clase de regla cuando sea necesario. El Editor CER proporciona el mecanismo de contención para diseñar la regla. Consulte la sección de clase de reglas y conjunto de reglas de cambio en “Asistentes de elemento de regla” en la página 114.

A veces puede ser complicado identificar qué mecanismo se debe utilizar al centralizar reglas comunes. En general, deberá utilizar la herencia cuidadosamente, sólo cuando la clase de subregla represente un concepto empresarial que verdaderamente *"es una"* instancia del concepto empresarial representado por la superclase. En concreto, CER no soporta la herencia múltiple.

Por ejemplo, una herencia se produce cuando una persona tiene recursos y cada recurso puede ser un edificio o un vehículo. Las clases de regla de edificio (`Building`) y vehículo (`Vehicle`) amplían cada una la clase de regla de recurso (`Resource`) abstracta. Consulte el listado en “Clases de regla” en la página 41.

Debe utilizar la contención cuando el concepto empresarial representado por una clase de regla *"tiene una"* instancia del concepto empresarial representado por la clase de regla que está contenida.

Por ejemplo, se produce una contención cuando a una persona se le aplican muchas pruebas de rango de edad diferentes. La clase de regla `Person` crea muchas instancias de `AgeRangeTest`.

Si encuentra que tiene clases de regla similares en diferentes conjuntos de reglas, deberá tener en cuenta la posibilidad de utilizar los recursos de CER (a partir de Cúram V6) para permitir que un conjunto de reglas haga referencia a artefactos de otro. Coloque las clases de regla comunes en uno o varios conjuntos de reglas comunes y ponga las clases de reglas que no son comunes en otros conjuntos de reglas.

Eliminar reglas no utilizadas

Al centralizar las reglas comunes, es posible que detecte que ya no se hace referencia a algunos atributos de regla desde otros cálculos y que dichos atributos se pueden eliminar del conjunto de reglas (como parte de un ejercicio de "orden").

CER incluye soporte para informar de atributos de regla a los que no se hace referencia desde ningún otro cálculo del conjunto de reglas y, por lo tanto, son candidatos para la eliminación. También puede utilizar el Editor CER para suprimir cualquier clase de regla y atributo de regla. Consulte “Vista técnica” en la página 105.

Para ejecutar el informe de atributo CER no utilizado, consulte “Atributos de regla no utilizados” en la página 29.

aviso: Tenga en cuenta que es perfectamente posible que un atributo de regla sea un atributo de regla de "nivel superior" al que sólo se hace referencia desde el código de cliente; estos atributos pueden ser tratados como "no utilizados" por este informe, pero los atributos de regla aparentemente no utilizados no deben eliminarse del conjunto de reglas a menos que esté seguro de que ningún código de cliente o pruebas dependen de ellos.

Orden de declaraciones

En general, el orden de declaración en los conjuntos de reglas no tiene ningún efecto en el comportamiento.

Orden de clases de reglas en un conjunto de reglas

Puede volver a ordenar las clases de regla del conjunto de reglas en el orden que le sea más útil. La reordenación de las clases de regla no tiene ningún efecto en el comportamiento de los conjuntos de reglas.

Orden de atributos de regla calculados en una clase de regla

Del mismo modo, puede reordenar los atributos de regla *calculados* de la clase de regla en el orden que le resulte más útil. La reordenación de atributos de regla *calculados* no tiene ningún efecto en el comportamiento de los conjuntos de reglas.

Orden de atributos inicializados en una clase de regla

Siempre que se crea una instancia de objeto de regla, ya sea en las reglas mediante el de la expresión `create`, ya sea a través de código Java utilizando las clases de regla generadas o la API de regla dinámica, los valores de todos los atributos inicializados se deben especificar *en el orden en que están definidos en la clase de regla*.

aviso: Por eso debe evitar reordenar los atributos de un bloque `Initialization` a menos que esté preparado para actualizar también todos los lugares (en las reglas o el código Java) que crean instancias de objeto de regla de la clase de regla.

Orden de condiciones booleanas

El orden de las condiciones booleanas en una expresión `all` o `any` no afecta al valor lógico del resultado.

Sin embargo, en tiempo de ejecución, el proceso de las condiciones booleanas se detiene tan pronto como se confirma un resultado; por eso, debe tener en cuenta el orden de las condiciones booleanas para que la expresión `all` o `any` tienda a obtener el resultado tan pronto como sea posible.

Esto significa:

- para las expresiones `all`, colocar las condiciones booleanas que probablemente se evaluarán como *false* primero en la lista y
- para las expresiones `any`, colocar las condiciones booleanas que probablemente se evaluarán como *true* primero en la lista.

Puede utilizar el Editor CER para cambiar el orden de los elementos de regla booleanos dentro del elemento de regla Any. Por ejemplo, organice todos los elementos de regla booleanos con el valor true en primer lugar.

Creación de objetos de regla

Debe examinar cuidadosamente cómo se crean los objetos de regla.

En concreto, si está utilizando CER en su propia aplicación, debe decidir pronto qué objetos de regla se crean mediante el código de aplicación o qué objetos de regla se crean mediante las reglas.

Consulte “Objetos de regla externa e interna” en la página 33 para obtener más detalles.

Pasar objetos de regla en vez de pasar los ID

Cuando utilice la expresión "`create`" en la página 183 para crear un objeto de regla interna, puede pasar datos al nuevo objeto de regla mediante el uso del bloque de inicialización y/o `especificar` elementos.

Si los datos que está pasando incluyen una referencia a los datos externos que tienen un identificador (p. ej. un caso, identificado por un caseID), considere la posibilidad de diseñar la clase de regla para el objeto de regla creado para que se inicialice con un objeto de regla que represente dichos datos, en lugar de inicializarse mediante un valor de ID.

El uso de un objeto de regla de este tipo para la inicialización puede aumentar la "seguridad de tipo" de los datos; puede ayudar a evitar que otro diseñador de reglas cree sus propios objetos de regla interna para la misma clase de regla, pero pasando accidentalmente un ID que representa una clase diferente de datos externos.

Es probable que pasar la clase incorrecta de ID haga que fallen las reglas en *tiempo de ejecución* (por ejemplo porque un intento de convertir un objeto de regla para ese ID no encontrará los datos subyacentes); mientras que pasar un objeto de regla para la seguridad de tipo permitirá que el validador de conjunto de reglas CER detecte el problema en el *momento de diseño*.

Desarrollo de métodos estáticos

CER tiene soporte para muchas expresiones que probablemente proporcionan los cálculos que necesita.

Si tiene un cálculo de negocio que no se puede implementar utilizando expresiones de CER, CER soporta la expresión `call` para permitirle llamar desde el conjunto de reglas a un método estático en una clase Java personalizada. El Editor CER proporciona pocos elementos de regla, por ejemplo "call" para permitir a los usuarios definir un método estático en una clase Java personalizada. Consulte el elemento de regla "call" en "Call" en la página 131.

A continuación se muestra un conjunto de reglas de ejemplo que llama a un método Java:

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_StaticMethodDevelopment"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Income">

    <Attribute name="paymentReceivedDate">
      <type>
        <javaclass name="curam.util.type.Date"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="amount">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
```

```
      </derivation>
    </Attribute>

  </Class>

  <Class name="Person">

    <Attribute name="incomes">
      <type>
        <javaclass name="List">
          <ruleclass name="Income"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="mostRecentIncome">
      <type>
        <ruleclass name="Income"/>
      </type>
      <derivation>
        <!-- En el primer desarrollo, se utilizaría el método
             identifyMostRecentIncome_StronglyTyped.

             En la práctica, no mantendrá dos versiones de
             cada método; simplemente debilitará los tipos de
             argumento y retorno y conservará un solo método.
          -->

        <call class="curam.creole.example.BestPracticeDevelopment"
          method="identifyMostRecentIncome_WeaklyTyped">
          <type>
            <ruleclass name="Income"/>
          </type>
          <arguments>
            <this/>
          </arguments>
        </call>
      </derivation>
    </Attribute>
  </Class>

</RuleSet>
```

Cuando desarrolle el método estático, puede empezar utilizando clases Java generadas de CER como tipos de argumento y/o tipo de retorno para el método:

```
package curam.creole.example;

import java.util.List;

import curam.creole.execution.session.Session;
import
 curam.creole.ruleclass.Example_StaticMethodDevelopment.impl.Income;
import
 curam.creole.ruleclass.Example_StaticMethodDevelopment.impl.Person;

public class BestPracticeDevelopment {

  /**
   * Identifica los ingresos más recientes de una persona.
   *
   * Tenga en cuenta que este cálculo puede realizarse utilizando
   * expresiones CER, pero se muestra aquí en Java sólo para ilustrar el
   * uso de tipos generados al desarrollar de métodos Java estáticos para
```

```
   * el uso con CER.
   *
   * Este método es adecuado sólo para su uso en el desarrollo; para
   * producción, consulte
   * {@linkplain #identifyMostRecentIncome_WeaklyTyped} más abajo.
   *
   * En la práctica, no mantendrá dos versiones de cada
   * método; simplemente debilitará los tipos de argumento y retorno
   * y mantendrá un solo método.
   */
  public static Income identifyMostRecentIncome_StronglyTyped(
      final Session session, final Person person) {

    Income mostRecentIncome = null;
    final List<? extends Income> incomes =
        person.incomes().getValue();
    for (final Income current : incomes) {
      if (mostRecentIncome == null
          || mostRecentIncome.paymentReceivedDate().getValue()
              .before(current.paymentReceivedDate().getValue())) {
        mostRecentIncome = current;
      }
    }

    return mostRecentIncome;
  }
}
```

Una vez que el método Java funciona de forma satisfactoria, debe debilitar los tipos para utilizar objetos de regla dinámicos en lugar de las clases Java generadas (que no deben utilizarse en un entorno de producción):

```
/**
   * Identifica los ingresos más recientes de una persona.
   *
   * Tenga en cuenta que este cálculo puede realizarse utilizando
   * expresiones CER, pero se muestra aquí en Java sólo para ilustrar el
   * uso de tipos generados al desarrollar de métodos Java estáticos para
   * el uso con CER.
   *
   * Este método es adecuado sólo para utilizarse en producción; para el
   * desarrollo inicial, consulte
   * {@linkplain #identifyMostRecentIncome_StronglyTyped} más arriba.
   *
   * En la práctica, no mantendrá dos versiones de cada
   * método; simplemente debilitará los tipos de argumento y retorno
   * y mantendrá un solo método.
   */
  public static RuleObject identifyMostRecentIncome_WeaklyTyped(
      final Session session, final RuleObject person) {

    RuleObject mostRecentIncome = null;
    final List<? extends RuleObject> incomes =
        (List<? extends RuleObject>) person.getAttributeValue(
            "incomes").getValue();
    for (final RuleObject current : incomes) {
      if (mostRecentIncome == null
          || ((Date) mostRecentIncome.getAttributeValue(
              "paymentReceivedDate").getValue())
              .before((Date) current.getAttributeValue(
                  "paymentReceivedDate").getValue())) {
        mostRecentIncome = current;
      }
    }
```

```
    return mostRecentIncome;
  }
```

PRECAUCIÓN:
Si cambia la implementación de un método estático, CER *no* sabrá automáticamente volver a calcular los valores de atributo que se han calculado utilizando la versión antigua del método estático.

Una vez que se ha utilizado un método estático en un entorno de producción para los valores de atributo almacenados, en lugar de cambiar la implementación deberá crear un método estático nuevo (con la nueva implementación necesaria) y cambiar los conjuntos de reglas para utilizar el método estático nuevo. Al publicar los cambios de conjunto de reglas para apuntar al nuevo método estático, CER volverá a calcular automáticamente todas las instancias de los valores de atributo afectados.

Cualquier método estático (o método de propiedad) invocado desde CER:

- *sólo* **debe depender de los datos que se le han pasado (es decir, su comportamiento debe ser determinista, basado en los parámetros de entrada). No debe obtener datos de otros orígenes como la base de datos, variables de entorno o globales variables, ya que si se obtienen significa que CER no puede detectar cuándo se han cambiado dichos datos y, por lo tanto, los recálculos dejarán de ser fiables y**
- *no* **debe realizar ningún efecto secundario (por ejemplo grabar datos en la base de datos), porque CER no garantiza el orden en el que se produce el proceso ni la frecuencia con la que se invocarán los métodos estáticos/de propiedad.**

Evitar inconvenientes comunes en las pruebas

Escribir pruebas JUnit para los conjuntos de reglas CER es muy sencillo, pero hay varios inconvenientes que debe conocer.

En esta sección se muestran algunos ejemplos de código de prueba que a primera vista parecen correctos, pero ninguno de ellos podrá producir el comportamiento necesario.

Evitar `assertEquals` de JUnit

Las pruebas de JUnit heredan métodos de aserción para pruebas. Normalmente se puede afirmar que un resultado *esperado* se compara con un resultado *real*.

Un inconveniente común es utilizar `assertEquals` de JUnit para encontrar que no funciona correctamente para las comparaciones numéricas (y con un mensaje de error ligeramente confuso para arrancar).

CER convierte todas las instancias de `Number` a su propio formato numérico (respaldado por `java.math.BigDecimal`) antes de procesarse, para asegurar que no se pierde precisión. Esta conversión puede ser problemática y no intuitiva si utiliza el método `assertEquals` de JUnit.

CER incluye una sustitución en una clase auxiliar. Utilice `CREOLETestHelper.assertEquals`, que comparará correctamente los números de cualquier tipo.

Para otros tipos de datos, `CREOLETestHelper.assertEquals` se comporta de forma idéntica a `assertEquals` de JUnit, por lo que, en general, es recomendable utilizar

CREOLETestHelper.assertEquals a lo largo de las pruebas, para evitar posibles confusiones (incluso en los lugares donde técnicamente es innecesario).

```
public void creoleTestHelperNotUsed() {

    final FlexibleRetirementYear flexibleRetirementYear =
        FlexibleRetirementYear_Factory.getFactory().newInstance(
            session);

    flexibleRetirementYear.retirementCause().specifyValue(
        "Se ha alcanzado la edad de jubilación legal.");

    /**
     * No funcionará -  getValue devuelve el propio manejador numérico de CER,
     * pero 65 es un entero.
     *
     * JUnit proporcionará el mensaje algo confuso:
     * junit.framework.AssertionFailedError: se esperaba:<65> pero
     * ha sido:<65>
     *
     * Utilice CREOLETestHelper.assertEquals en su lugar.
     */
    assertEquals(65, flexibleRetirementYear.ageAtRetirement()
        .getValue());

  }
```

Recordar utilizar .getValue()

El generador de códigos de prueba de CER crea una interfaz de Java para cada clase de regla y un método de descriptor de acceso en la interfaz para cada atributo de regla.

Este método de descriptor de acceso generado devuelve un AttributeValue de CER, *no* el valor del atributo directamente. Para obtener el propio valor, debe llamar al método .getValue() en AttributeValue.

Si se olvida de utilizar .getValue() en una prueba, probablemente la prueba se compilará de manera adecuada pero no podrá comportarse correctamente cuando se ejecute.

```
public void getValueNotUsed() {

    final FlexibleRetirementYear flexibleRetirementYear =
        FlexibleRetirementYear_Factory.getFactory().newInstance(
            session);

    flexibleRetirementYear.retirementCause().specifyValue(
        "Se ha alcanzado la edad de jubilación legal.");

    /**
     * No funcionará - ageAtRetirement() es una calculadora, no un
     * valor.
     *
     * JUnit proporcionará el mensaje:
     * junit.framework.AssertionFailedError: se esperaba:<65> pero
     * ha sido: <Valor: 65>
     *
     * Recuerde que debe utilizar .getValue() en cada calculadora de atributo.
     */
    assertEquals(65, flexibleRetirementYear.ageAtRetirement());

  }
```

Tenga en cuenta que en este ejemplo, el valor del AttributeValue se muestra como la serie "Valor: 65", en lugar del número 65 (que es lo .getValue() habría devuelto).

Recordar que se deben especificar todos los valores necesarios para los cálculos que se están probando

En las pruebas, sólo necesita especificar los valores a los que se accederá durante la ejecución de las reglas.

Sin embargo, puede ser fácil olvidarse de especificar un valor; si es así, cuando CER intenta un cálculo, puede encontrar un atributo cuya derivación es <specified>, pero para el que no se ha especificado ningún valor en el código de prueba y CER informará de una pila de errores:

```
public void valueNotSpecified() {

    final FlexibleRetirementYear flexibleRetirementYear =
        FlexibleRetirementYear_Factory.getFactory().newInstance(
            session);

    /**
     * No funcionará - un valor necesario para el cálculo se ha marcado
     * como <specified> pero no se ha especificado ningún valor para el mismo.
     *
     * CER emitirá una pila de mensajes:
     * <ul>
     *
     * <li> Error al calcular atributo 'ageAtRetirement' en regla
     * class 'FlexibleRetirementYear' (instance id '1', description
     * 'Instancia no descrita de la clase de regla
     * 'FlexibleRetirementYear', id '1'). </li>
     *
     * <li>Error al calcular el atributo 'retirementCause' en la regla
     * class 'FlexibleRetirementYear' (instance id '1', description
     * 'Instancia no descrita de la clase de regla
     * 'FlexibleRetirementYear', id '1'). </li>
     *
     * <li>El valor se debe especificar antes de utilizarse (no se puede
     * calcular).</li>
     *
     * </ul>
     *
     *
     * Recuerde que debe especificar todos los valores necesarios para los cálculos.
     */
    CREOLETestHelper.assertEquals(65, flexibleRetirementYear
        .ageAtRetirement().getValue());

  }
```

No especificar el mismo valor más de una vez

CER le permite especificar un valor que de lo contrario se calcularía.

Sin embargo, cuando se utiliza la estrategia RecalculationsProhibited, CER generará un error de tiempo de ejecución si intenta especificar el valor de un atributo (en un objeto de regla determinado) más de una vez; una vez que el valor se ha especificado, no se puede cambiar (porque hacerlo puede significar que los cálculos realizados anteriormente son ahora “incorrectos”).

```
public void valueSpecifiedTwice() {

    final FlexibleRetirementYear flexibleRetirementYear =
        FlexibleRetirementYear_Factory.getFactory().newInstance(
            session);
```

```
    flexibleRetirementYear.retirementCause().specifyValue(
        "Se ha alcanzado la edad de jubilación legal.");

    /**
     * No funcionará - el mismo valor de atributo no se puede especificar
     * una segunda vez.
     *
     * CER informará del mensaje: No se puede especificar un valor,
     * porque el estado actual de esta calculadora es 'SPECIFIED'.
     *
     * No intente especificar el mismo valor dos veces.
     */
    flexibleRetirementYear.retirementCause().specifyValue(
        "Ganador de lotería");

  }
```

Especificar el tipo correcto de valor para un atributo

Cada `AttributeValue` de CER tiene un método `specifyValue` para permitir que se especifique el valor del atributo (en lugar de calcularlo). El método adopta cualquier valor, pero si especifica el tipo de valor incorrecto, CER generará un error de tiempo de ejecución tal como se indica:

```
public void incorrectValueType() {

    final FlexibleRetirementYear flexibleRetirementYear =
        FlexibleRetirementYear_Factory.getFactory().newInstance(
            session);

    /**
     * No funcionará - retirementCause() espera una serie, no un
     * número.
     *
     * CER mostrará el mensaje: Intento de establecer el valor '123'
     * (de tipo 'java.lang.Integer') en el atributo 'retirementCause'
     * de la clase de regla 'FlexibleRetirementYear' (que espera un
     * 'java.lang.String').
     */
    flexibleRetirementYear.retirementCause().specifyValue(123);

  }
```

Nota:

Los lectores técnicos podrían preguntarse por qué `AttributeValue.specifyValue` no utiliza genéricos de Java 5 para restringir el tipo de valor que puede recibir.

Si una clase de regla A amplía la clase de regla B, A puede alterar temporalmente la derivación de cualquiera de los atributos de B. A también puede volver a declarar cualquiera de los atributos de B para que sean de un tipo más restrictivo (es decir, la declaración de A del atributo especifica que su tipo sea un subtipo del declarado por B).

La interfaz Java generada para A amplía la interfaz Java generada para B. Dado que los descriptores de acceso devuelven calculadoras, en lugar del tipo de valor directamente, todas las interfaces deben utilizar la extensión de comodín para que el compilador permita que la declaración de A del descriptor de acceso del atributo amplíe la de B. Dado que se utiliza la extensión comodín, `specifyValue` no puede restringirse a un tipo y, por lo tanto, se debe declarar para recibir cualquier objeto (`Object`).

Desde otro punto de vista, si una clase Java C tuviera que ampliar Java D, C puede definir un tipo de retorno más restrictivo para uno de los métodos get de D, pero no puede restringir los métodos set de D a un subtipo: C debe implementar el método de D y detectar los valores no deseados en el tiempo de ejecución (aunque al hacerlo es posible que viole de manera cuestionable el principio de sustitución de Liskov).

Otra justificación es que, cuando se utilizan los objetos de regla puramente-dinámicos (es decir, en una sesión interpretada), la restricción de tiempo de compilación de valores no se puede utilizar.

Por lo tanto, CER utiliza su conocimiento de tipos de atributo declarados para detectar valores incorrectos en el tiempo de ejecución, no el tiempo de compilación.

Crear todos los objetos de regla para una sesión antes de ejecutar cálculos `getValue`

Las pruebas de conjunto de reglas pueden configurar cualquier número de objetos de regla en una sesión CER antes de continuar con la comprobación de cualquier número de valores calculados en esos objetos de regla.

Sin embargo, una vez que se hayan comenzado los cálculos, la estrategia `RecalculationsProhibited` impide crear objetos de regla que invaliden el valor de un cálculo "`readall`" en la página 216 previamente calculado.

Debe estructurar las pruebas de modo que la creación de todos los objetos de regla de prueba se produzca *antes de* cualquier cálculo (es decir, antes de cualquier ejecución de `getValue`). En la práctica esto no es excesivamente restrictivo.

Si la prueba intenta crear un objeto de regla nuevo en una sesión después de que se haya producido un cálculo, (si los cálculos `readall` calculados anteriormente han quedado afectados), la estrategia `RecalculationsProhibited` generará un error de tiempo de ejecución:

```
public void newObjectsAddedAfterCalculationsStarted() {

    final FlexibleRetirementYear flexibleRetirementYear =
        FlexibleRetirementYear_Factory.getFactory().newInstance(
            session);

    flexibleRetirementYear.retirementCause().specifyValue(
        "Se ha alcanzado la edad de jubilación legal.");

    /**
     * Calcular la edad de jubilación y probar su valor
     */
    CREOLETestHelper.assertEquals(65, flexibleRetirementYear
        .ageAtRetirement().getValue());

    /**
     * Crear otro objeto de regla.
     */

    /**
     * Puede que no funcione: los objetos de regla nuevos añadidos a la sesión una vez que
     * se han iniciado los cálculos pueden invalidar los cálculos
     * <code><readall></code> anteriores.
     *
     * {@linkplain RecalculationsProhibited} puede generar el
     * mensaje: "No se pueden crear objetos de regla nuevos para esta sesión,
     * porque esta sesión ya ha aceptado una solicitud
     * de cálculo."
     *
```

```
     * Para evitar este problema, cree todos los objetos de regla antes
     * de intentar cualquier cálculo.
     */
    final FlexibleRetirementYear flexibleRetirementYear2 =
        FlexibleRetirementYear_Factory.getFactory().newInstance(
            session);

  }
```

aviso: Si el conjunto de reglas no contiene *actualmente* ninguna expresión `readall`, puede no estructurar las pruebas para que toda la creación de objetos de regla se produzca antes de cualquier cálculo.

Sin embargo, si cambia el conjunto de reglas en el futuro para que contenga expresiones `readall`, tendrá que reestructurar sus pruebas en ese momento.

Para evitar el reacondicionamiento, estructure siempre las pruebas para que toda la creación de objetos de regla se realice antes de que empiecen los cálculos.

Diccionario XML de CER

Una descripción de los elementos que conforman el idioma de CER para los conjuntos de reglas.

Conjunto de reglas

Un conjunto de reglas especifica su *nombre* y contiene cualquier número de `Clases` de regla y/o sentencias `Include`. Opcionalmente el conjunto de reglas puede contener `Anotaciones`.

La estructura XML de un conjunto de reglas y sus elementos está restringido por el esquema `RuleSet.xsd` de CER. Este esquema se construye dinámicamente, para que las ampliaciones en CER puedan contribuir con expresiones y anotaciones en el esquema.

A continuación se muestra un ejemplo de esquema de un conjunto de reglas:

```
RuleSet
  Annotations (opcional)
    ...
    ...
  Include
    ...
  Include
    ...
  ... more Include statements
  Clase
    Annotations (opcional)
      ...
      ...
  Initialization (opcional)
      Attribute
        Annotations (opcional)
          ...
          ...
      type
          ...
      Attribute
        type
          ...
      ... more initialized attributes
    Atributo
      Annotations (opcional)
```

```
      ...
      ...
  type
      ...
    derivation
      (expresión)
        Annotations (opcional)
          ...
          ...
        (subexpresión)
        ...
  Attribute
    type
      ...
    derivation
      ...
    ... more calculated attributes
... more rule classes
```

Sentencia `Include`

Es posible que le resulte práctico dividir un gran conjunto de reglas en partes más pequeñas para facilitar el desarrollo o la reutilización paralelos. Cada conjunto de reglas puede contener sentencias `Include` para incluir otros conjuntos de reglas y clases. El elemento raíz en un elemento incluido debe ser:

- **Class**

  Una clase de regla única o

- **RuleSet**

  Un conjunto de reglas entero, que él mismo puede contener sus propias sentencias `Include` que se procesarán de forma repetida.

Se soportan diferentes tipos de sentencias `Include`:

- **RelativePath**

  Incluye un archivo XML con una vía de acceso relativa al archivo que lo contiene; este mecanismo puede ser útil durante el desarrollo autónomo del conjunto de reglas por parte de los desarrolladores que utilizan un entorno de desarrollo basado en archivo;

- **Classpath**

  Incluye un archivo XML que está presente en la ubicación mencionada en la vía de acceso de clases de tiempo de ejecución; este mecanismo puede ser útil para hacer referencia a conjuntos de reglas comunes que raramente cambian y se crean en la aplicación.

Consejo: Dentro de un conjunto de reglas, no hay ningún significado asociado al orden en que se especifican las sentencias `Include` Puede volver a ordenar las sentencias `Include` en un conjunto de reglas sin que ello afecte al comportamiento del conjunto de reglas.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_Include"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">

  <!-- Esta clase de regla se define directamente en este conjunto de reglas -->
  <Class name="Person">
    <Attribute name="firstName">
      <type>
        <javaclass name="String"/>
      </type>
```

```
      <derivation>
        <specified/>
      </derivation>
    </Attribute>
  </Class>

  <!-- Incluir un conjunto de reglas definido en otro archivo.

       Cuando se ensamblan en un solo conjunto de reglas, los
       nombres de todas las clases de regla deben ser exclusivos. -->
  <Include>
    <RelativePath value="./HelloWorld.xml"/>
  </Include>

</RuleSet>
```

Consulte “Consolidador de conjunto de reglas CER” en la página 29 para saber cómo contraer un conjunto de reglas que contiene inclusiones `RelativePath` en un solo archivo de conjunto de reglas.

Clase de regla

Una clase de regla define el comportamiento de las instancias de objeto de regla.

Una clase de regla especifica su *nombre* (que debe ser exclusivo entre las clases de reglas en el conjunto de reglas), indica si es *abstracta* y contiene:

- **Initialization**

  Opcionalmente, un bloque de atributos en el que se deben especificar los valores siempre que se crea una instancia de objeto de regla de la clase y
- **Attribute**

  Cero o más sentencias `Attribute` calculadas, cada una de las cuales describe un valor que la clase de regla pueda calcular.

Una clase de regla se puede definir en su propio archivo XML (es decir, donde el elemento XML raíz es `Class`) e incluir en un conjunto de reglas padre utilizando una sentencia “Sentencia `Include`” en la página 159.

Atributos inicializados

El bloque `Initialization` contiene uno o más sentencias `Attribute`, cada una de las cuales especifica el `type` del atributo, pero *no* especifica ninguna `derivation`.

Siempre que se crea una instancia de objeto de regla, ya sea en las reglas mediante el de la expresión `create`, ya sea a través de código Java utilizando las clases de regla generadas o la API de regla dinámica, los valores de todos los atributos inicializados se deben especificar *en el orden en que están definidos en la clase de regla*.

aviso: Por eso debe evitar reordenar los atributos de un bloque `Initialization` a menos que esté preparado para actualizar también todos los lugares (en las reglas o el código Java) que crean instancias de objeto de regla de la clase de regla.

Atributos calculados

Los atributos calculados se listan directamente en la clase de regla.

Consejo: En una clase de regla, no hay ningún significado asociado al orden en que se especifican los atributos calculados. Puede volver a ordenar los atributos calculados en una clase de regla sin que ello afecte al comportamiento de la clase de regla.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_RuleClass"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Initialization>
      <!-- Cada atributo inicializado contiene un tipo
           pero ninguna derivación.

           NO debe volver a ordenar de forma arbitraria
           los atributos inicializados. -->
      <Attribute name="firstName">
        <type>
          <javaclass name="String"/>
        </type>
      </Attribute>
      <Attribute name="age">
        <type>
          <javaclass name="Number"/>
        </type>
      </Attribute>
    </Initialization>

    <!-- Cada atributo calculado especifica
         un tipo y una derivación.

         Puede volver a ordenar arbitrariamente
         los atributos calculados. -->
    <Attribute name="isAdult">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <compare comparison=">=">
          <reference attribute="age"/>
          <Number value="18"/>
        </compare>
      </derivation>
    </Attribute>

    <Attribute name="isSeniorCitizen">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <compare comparison=">=">
          <reference attribute="age"/>
          <Number value="65"/>
        </compare>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

Atributo

Cada atributo especifica su *nombre* (que debe ser exclusivo entre los atributos de regla definidos en la clase de regla o heredados de la misma) y contiene:

Cada `Attribute` contiene:

- **type**

Define el tipo de valor que este atributo proporciona (consulte “Tipos de datos soportados” en la página 41) y

- **derivation (sólo atributos calculados)**

  Define cómo el atributo calcula su valor. Cada `derivation` contiene una sola expresión CER (consulte “Listado alfabético completo de expresiones” en la página 164).

  Importante: Existen expresiones de marcador especiales que pueden modificar la semántica del atributo de regla; consulte “Marcadores” en la página 164.

Expresiones

CER soporta una amplia gama de expresiones. Las expresiones se listan alfabéticamente más abajo, pero también se agrupan lógicamente aquí para consulta (tenga en cuenta que algunas expresiones aparecen de forma intencionada en más de un grupo lógico).

Lógica booleana

- “`true`” en la página 246
- “`false`” en la página 196
- “`all`” en la página 168
- “`any`” en la página 170
- “`not`” en la página 208

Comparación de valores

- “`equals`” en la página 192
- “`compare`” en la página 181
- “`sort`” en la página 230

Constantes

Estas expresiones proporcionan valores constantes literales.

- “`true`” en la página 246
- “`false`” en la página 196
- “`null`” en la página 208
- “`String`” en la página 232
- “`Número`” en la página 210
- “`Date`” en la página 188
- “`Code`” en la página 180
- “`FrequencyPattern`” en la página 201

Lógica condicional

- “`choose`” en la página 177

Agregaciones de lista

Estas expresiones agregan una lista de valores a un valor derivado.

- “`all`” en la página 168
- “`any`” en la página 170
- “`sum`” en la página 234
- “`min`” en la página 206
- “`max`” en la página 204
- “`concat`” en la página 182

- "singleitem" en la página 228

Para obtener operaciones adicionales proporcionadas directamente desde la interfaz de Java java.util.List, consulte "Operaciones de lista útiles" en la página 252.

Transformaciones de lista

Estas expresiones transforman una lista para crear una lista nueva.

- "dynamiclist" en la página 189
- "fixedlist" en la página 199
- "filter" en la página 196
- "joinlists" en la página 203
- "removeduplicates" en la página 224
- "sort" en la página 230
- "sublists" en la página 233

Mensajes traducibles

Estas expresiones permiten la creación de mensajes que se pueden visualizar en el idioma/la configuración regional del usuario final.

- "concat" en la página 182
- "ResourceMessage" en la página 226
- "XmlMessage" en la página 246

Cálculos numéricos

Estas expresiones soportan los cálculos numéricos:

- "Número" en la página 210
- "arithmetic" en la página 173
- "periodlength" en la página 211
- "sum" en la página 234
- "max" en la página 204
- "min" en la página 206

Referencias

Estas expresiones permiten que un cálculo haga referencia a otro elemento.

- "reference" en la página 221
- "current" en la página 186
- "this" en la página 236

Creation

Esta expresión permite la creación de un nuevo objeto de regla.

- "create" en la página 183

Recuperación

Esta expresión permite la recuperación de objetos de regla.

- "readall" en la página 216

Comentarios emergentes de Java

Estas expresiones permiten la invocación de código Java para realizar un cálculo.

- "property" en la página 213
- "call" en la página 175

Marcadores

Estas expresiones son marcadores especiales (no son cálculos como tales).

- "abstract"
- "specified" en la página 232

Líneas de tiempo

Estas expresiones procesan líneas de tiempo de CER.

Para obtener más detalles sobre las líneas de tiempo de CER, consulte "Manejo de datos que cambian a lo largo del tiempo" en la página 51 en esta guía.

- "Intervalo" en la página 202
- "Timeline" en la página 237
- "existencetimeline" en la página 194
- "intervalvalue" en la página 203
- "timelineoperation" en la página 239

Elegibilidad y titularidad de entrega de productos

Estas expresiones proporcionan cálculos específicos de empresa para la elegibilidad y titularidad de casos de entrega de productos.

Consulte la guía Inside Cúram Eligibility and Entitlement Using Cúram Express Rules para obtener una descripción de estas expresiones.

- "combineSuccessionSets" en la página 181
- "legislationChange" en la página 204
- "rate" en la página 216

Listado alfabético completo de expresiones

Esta sección define todas las expresiones incluidas con CER que se incluyen con la aplicación.

Las expresiones siguientes se listan alfabéticamente; consulte los apartados anteriores para obtener categorizaciones útiles de esas expresiones.

Nota: Algunas expresiones son para las derivaciones específicas de empresa en la aplicación. Las expresiones de este tipo todavía se incluyen aquí, pero se remite al lector a otras guías de Cúram que describen esas expresiones en su contexto empresarial.

Para ser breves, los conjuntos de reglas de ejemplo se muestran sin anotaciones; en la práctica, los conjuntos de reglas guardados utilizando el Editor CER contendrán anotaciones para información de diagrama y descripción.

abstract:
Expresión de marcador para indicar que la derivación del atributo debe especificarse en subclases concretas (o una de las superclases).

Si uno o más atributos en una clase de regla está marcado abstract, el validador de conjunto de reglas CER insistirá en que la propia clase se marque *abstract="true"* e impedirá que la clase de regla se utilice en cualquier expresión de "create" en la página 183.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_abstract"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
```

```
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">

  <!-- Clase base para todos los tipos de prestación.
       Cada subclase concreta tiene su propio
       cálculo de "name" y "isEligible". -->
  <Class name="Benefit" abstract="true">
    <Initialization>
      <!-- La persona para la que se está
        determinando la elegibilidad de prestación. -->
      <Attribute name="person">
        <type>
          <ruleclass name="Person"/>
        </type>
      </Attribute>
    </Initialization>


    <!-- El nombre de este tipo es prestación -->
    <Attribute name="name">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <abstract/>
      </derivation>
    </Attribute>

    <!-- Indica si la persona es elegible para esta prestación. -->
    <Attribute name="isEligible">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <abstract/>
      </derivation>
    </Attribute>

  </Class>

  <!-- Una subclase concreta de prestación.
       Contiene derivaciones concretas para los atributos
       abstractos heredados. -->
  <Class name="MedicalBenefit" extends="Benefit">
    <Attribute name="name">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <String value="Prestación médica"/>
      </derivation>
    </Attribute>
    <Attribute name="isEligible">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <all>
          <fixedlist>
            <listof>
              <javaclass name="Boolean"/>
            </listof>
            <members>
              <!-- NB el atributo de persona se hereda de prestación
 -->
              <reference attribute="isPoor">
                <reference attribute="person"/>
              </reference>
```

```
            <reference attribute="isSick">
              <reference attribute="person"/>
            </reference>
          </members>
        </fixedlist>

      </all>

    </derivation>
  </Attribute>
</Class>


<!-- Otra subclase concreta de prestación,
  con derivaciones concretas diferentes para los atributos
  abstractos heredados. -->
<Class name="NeedyBenefit" extends="Benefit">
  <Attribute name="name">
    <type>
      <javaclass name="String"/>
    </type>
    <derivation>
      <String value="Prestación médica"/>
    </derivation>
  </Attribute>
  <Attribute name="isEligible">
    <type>
      <javaclass name="Boolean"/>
    </type>
    <derivation>
      <all>
        <fixedlist>
          <listof>
            <javaclass name="Boolean"/>
          </listof>
          <members>
            <reference attribute="isPoor">
              <reference attribute="person"/>
            </reference>
            <any>
              <fixedlist>
                <listof>
                  <javaclass name="Boolean"/>
                </listof>
                <members>
                  <reference attribute="isHungry">
                    <reference attribute="person"/>
                  </reference>
                  <reference attribute="isDeprived">
                    <reference attribute="person"/>
                  </reference>
                </members>
              </fixedlist>
            </any>
          </members>
        </fixedlist>

      </all>

    </derivation>
  </Attribute>
</Class>

<Class name="Person">

  <Attribute name="isPoor">
    <type>
```

```
    <javaclass name="Boolean"/>
  </type>
  <derivation>
    <specified/>
  </derivation>
</Attribute>

<Attribute name="isSick">
  <type>
    <javaclass name="Boolean"/>
  </type>
  <derivation>
    <specified/>
  </derivation>
</Attribute>

<Attribute name="isHungry">
  <type>
    <javaclass name="Boolean"/>
  </type>
  <derivation>
    <specified/>
  </derivation>
</Attribute>

<Attribute name="isDeprived">
  <type>
    <javaclass name="Boolean"/>
  </type>
  <derivation>
    <specified/>
  </derivation>
</Attribute>


<!-- Una lista de todas las prestaciones para
     las que se está evaluando la persona.  -->
<Attribute name="allBenefits">
  <type>
    <javaclass name="List">
      <ruleclass name="Benefit"/>
    </javaclass>
  </type>
  <derivation>
    <fixedlist>
      <listof>
        <ruleclass name="Benefit"/>
      </listof>
      <members>
        <!-- Crear instancias de la clases de regla concretas -->
        <create ruleclass="MedicalBenefit">
          <this/>
        </create>
        <create ruleclass="NeedyBenefit">
          <this/>
        </create>
      </members>
    </fixedlist>

  </derivation>
</Attribute>

<!-- Las prestaciones para las que esta persona
     es elegible.

     Tenga en cuenta que la lista es de la clase de
     regla abstracta "Benefit", pero que cada
```

```
        instancia concreta determina su
        elegibilidad de su propia manera.    -->
    <Attribute name="eligibleBenefits">
      <type>
        <javaclass name="List">
          <ruleclass name="Benefit"/>
        </javaclass>
      </type>
      <derivation>
        <filter>
          <list>
            <reference attribute="allBenefits"/>
          </list>
          <listitemexpression>
            <reference attribute="isEligible">
              <current/>
            </reference>
          </listitemexpression>
        </filter>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

all:
Opera en una lista de valores booleanos para determinar si todos los valores de lista son *true*.

El cálculo se detiene en el primer valor *false* que se encuentra en la lista. Si la lista está vacía, esta expresión devuelve *true*.

La lista de valores booleanos la proporciona normalmente proporcionado por una "fixedlist" en la página 199 o "dynamiclist" en la página 189.

Consejo: El orden de los elementos de la lista no cambia nada en el valor de esta expresión; sin embargo, por razones de rendimiento, es posible que desee estructurar una "fixedlist" en la página 199 para que los valores "fail fast" estén más cerca de la parte superior de la lista y los valores que pueden ser difíciles de calcular estén más cerca de la parte inferior de la lista.

Nota: Desde Cúram V6, CER ya no informa de errores en expresiones hijo en situaciones donde el error no afecta al resultado general.

Por ejemplo, si una lista fija de tres atributos booleanos tienen estos valores:

- `true`;
- <error durante el cálculo> y
- `false`

el cálculo del valor de `all` de estos valores devolverá `false`, porque como mínimo uno de los elementos es falso (false) (concretamente el tercero de la lista), independientemente de que el segundo elemento devuelva un error.

Por el contrario, si otra lista fija de tres atributos booleanos tiene estos valores:

- `true`;
- <error durante el cálculo> y
- `true`

el cálculo del valor de all para estos valores devolverá el error indicado por el segundo elemento de la lista, porque este error evita que se determine si todos los elementos tienen el valor true.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_all"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="isLoneParent">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <!-- Ejemplo de <all> operando en una <fixedlist> -->
        <!-- Para considerarse como un "progenitor solo", una persona debe
             no estar casada y tener como mínimo un hijo -->
        <all>
          <fixedlist>
            <listof>
              <javaclass name="Boolean"/>
            </listof>
            <members>
              <!-- Sabemos que la mayoría de la gente de nuestra
 base de datos
                   está casada, por lo tanto probamos esta condición primero.

                   Si lo que sucede es que el valor isMarried no se ha
                   especificado para una persona, si esa persona
                   no tiene hijos, <all> devolverá false;
                   de lo contrario devolverá un error indicando que
                   el valor de isMarried no se ha especificado.
                   -->
              <not>
                <reference attribute="isMarried"/>
              </not>
              <not>
                <property name="isEmpty">
                  <object>
                    <reference attribute="children"/>
                  </object>
                </property>
              </not>
            </members>
          </fixedlist>
        </all>
      </derivation>
    </Attribute>

    <Attribute name="hasNoYoungChildren">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <!-- Ejemplo de <all> operando en una <dynamiclist>.

             Si lo que sucede es que la edad de un hijo no se puede
             calcular y hay al menos un hijo de menos de 5 años de edad,
             <all> devolverá false; de lo contrario,
             devolverá el error que muestra por qué no se ha podido
             calcular la edad del hijo.
        -->

        <!-- Compruebe si los niños tienen todos más de 5 años de edad
-->
```

```
        <all>
          <dynamiclist>
            <list>
              <reference attribute="children"/>
            </list>
            <listitemexpression>
              <compare comparison="&gt;">
                <reference attribute="age">
                  <current/>
                </reference>
                <Number value="5"/>
              </compare>
            </listitemexpression>
          </dynamiclist>
        </all>
      </derivation>
    </Attribute>

    <!-- Los hijos de esta persona: cada hijo también es una persona.
 -->
    <Attribute name="children">
      <type>
        <javaclass name="List">
          <ruleclass name="Person"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="isMarried">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="age">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

any:
Opera en una lista de valores booleanos para determinar si cualquiera de los valores de listas es *true*.

El cálculo se detiene en el primer valor *true* que se encuentra en la lista. Si la lista está vacía, esta expresión devuelve *false*.

La lista de valores booleanos la proporciona normalmente proporcionado por una "fixedlist" en la página 199 o "dynamiclist" en la página 189.

Consejo: El orden de los elementos de la lista no cambia nada en el valor de esta expresión; sin embargo, por razones de rendimiento, es posible que desee

estructurar una “fixedlist” en la página 199 para que los valores “succeed fast” estén más cerca de la parte superior de la lista y los valores que pueden ser difíciles de calcular estén más cerca de la parte inferior de la lista.

Nota: Desde Cúram V6, CER ya no informa de errores en expresiones hijo en situaciones donde el error no afecta al resultado general.

Por ejemplo, si una lista fija de tres atributos booleanos tienen estos valores:

- false;
- <error durante el cálculo> y
- true

el cálculo del valor de cualquiera any de estos valores devolverá true, porque como mínimo uno de los elementos es verdadero (true) (concretamente el tercero de la lista), independientemente de que el segundo elemento devuelva un error.

Por el contrario, si otra lista fija de tres atributos booleanos tiene estos valores:

- false;
- <error durante el cálculo> y
- false

el cálculo del valor de any para estos valores devolverá el error indicado por el segundo elemento de la lista, porque este error impide determinar si cualquier elemento tiene el valor true.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_any"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="qualifiesForFreeTravelPass">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <!-- Ejemplo de <any> operando en una <fixedlist> -->
        <!-- Para tener derecho a un billete gratuito, la persona
             debe ser de la tercera edad, estar ciega o discapacitada -->
        <any>
          <fixedlist>
            <listof>
              <javaclass name="Boolean"/>
            </listof>
            <members>
              <!-- Sabemos que la mayoría de la gente de nuestra
                   base de datos son ciudadanos de la tercera edad, de modo que probamos
                   primero esta condición.

                   Si lo que sucede es que el valor isBlind no se ha
                   especificado para una persona, si esa persona es
                   discapacitada, <any> devolverá false;
                   de lo contrario devolverá un error indicando que
                   el valor de isBlind no se ha especificado.
                   -->

              <compare comparison="&gt;=">
                <reference attribute="age"/>
                <Number value="65"/>
              </compare>
```

```
            <reference attribute="isBlind"/>
            <reference attribute="isDisabled"/>
          </members>
        </fixedlist>
      </any>
    </derivation>
  </Attribute>

  <Attribute name="qualifiesForChildBenefit">
    <type>
      <javaclass name="Boolean"/>
    </type>
    <derivation>
      <!-- Ejemplo de <any> operando en una <dynamiclist>.

           Si lo que sucede es que la edad de un hijo no se puede
           calcular y hay al menos un hijo de menos de 16,
           <any> devolverá true; de lo contrario,
           devolverá el error que muestra por qué no se ha podido
           calcular la edad del hijo.

      -->
      <!-- Para tener derecho a la prestación por hijos, esta persona debe
           tener uno o más hijos de menos de 16 años de edad. -->
      <any>
        <dynamiclist>
          <list>
            <reference attribute="children"/>
          </list>
          <listitemexpression>
            <compare comparison="&lt;">
              <reference attribute="age">
                <current/>
              </reference>
              <Number value="16"/>
            </compare>
          </listitemexpression>
        </dynamiclist>
      </any>
    </derivation>
  </Attribute>

  <!-- Los hijos de esta persona: cada hijo también es una persona.
-->
  <Attribute name="children">
    <type>
      <javaclass name="List">
        <ruleclass name="Person"/>
      </javaclass>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

  <Attribute name="isBlind">
    <type>
      <javaclass name="Boolean"/>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

  <Attribute name="isDisabled">
    <type>
      <javaclass name="Boolean"/>
```

```
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="age">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

arithmetic:
Realiza un cálculo aritmético de dos números (un número del lado izquierdo y un número del lado derecho) y opcionalmente redondea el resultado al número especificado de decimales.

Las operaciones soportadas son:

- **Suma**

  Lado izquierdo + lado derecho;
- **Resta**

  Lado izquierdo - lado derecho;
- **Multiplicación**

  lado izquierdo * lado derecho; y
- **División**

  lado izquierdo / lado derecho.

Si es necesario redondear, debe especificar:

- El número de decimales al que se debe redondear y
- la modalidad de redondeo, es decir, la dirección en la que se debe redondear cuando se realiza el redondeo. Consulte el JavaDoc para `RoundingMode` para obtener la lista de modalidades de redondeo soportadas y una explicación detallada del comportamiento.

aviso: Para las operaciones de división, en general se debe proporcionar una modalidad de redondeo y un número de decimales. Si no se proporciona y no se puede calcular un resultado exacto en el tiempo de ejecución, se producirá un error de tiempo de ejecución.

El validador de conjunto de reglas CER emitirá un aviso si detecta cualquier operación de división en el conjunto de reglas que no especifica redondeo.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_arithmetic"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="ArithmeticExampleRuleClass">

    <!-- 3 + 2 = 5 -->
    <Attribute name="addANumberToAnother">
      <type>
```

```
    <javaclass name="Number"/>
  </type>
  <derivation>
    <arithmetic operation="+">
      <Number value="3"/>
      <Number value="2"/>
    </arithmetic>
  </derivation>
</Attribute>

<!-- 3 - 2 = 1 -->
<Attribute name="subtractANumberFromAnother">
  <type>
    <javaclass name="Number"/>
  </type>
  <derivation>
    <arithmetic operation="-">
      <Number value="3"/>
      <Number value="2"/>
    </arithmetic>
  </derivation>
</Attribute>

<!-- 3 * 2 = 6 -->
<Attribute name="multiplyANumberByAnother">
  <type>
    <javaclass name="Number"/>
  </type>
  <derivation>
    <arithmetic operation="*">
      <Number value="3"/>
      <Number value="2"/>
    </arithmetic>
  </derivation>
</Attribute>

<!-- 3 / 2 = 1.5 -->
<!-- Puesto que la división es por 2,
     podemos salir sin redondeo.
     De todas maneras el validador de conjunto
     de reglas CER emitirá un aviso.   -->
<Attribute name="divideANumbersByAnother">
  <type>
    <javaclass name="Number"/>
  </type>
  <derivation>
    <arithmetic operation="/">
      <Number value="3"/>
      <Number value="2"/>
    </arithmetic>
  </derivation>
</Attribute>

<!-- (3 + 2) * 4 = 20 -->
<Attribute name="chainedArithmetic">
  <type>
    <javaclass name="Number"/>
  </type>
  <derivation>
    <arithmetic operation="*">
      <arithmetic operation="+">
        <Number value="3"/>
        <Number value="2"/>
      </arithmetic>
      <Number value="4"/>
    </arithmetic>
  </derivation>
```

```
    </Attribute>

    <!-- 1,23 + 3,45 = 4,68,
         = 4,7 cuando se redondea al decimal más cercano-->
    <Attribute name="roundedAddition">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <arithmetic decimalPlaces="1" operation="+"
          rounding="half_up">
          <Number value="1.23"/>
          <Number value="3.45"/>
        </arithmetic>
      </derivation>
    </Attribute>

    <!-- 2 / 3, = 0,667 a 3 decimales -->
    <!-- Si no se especifica redondeo,
         se producirá un error de tiempo de ejecución -->
    <Attribute name="roundedDivision">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <arithmetic decimalPlaces="3" operation="/"
          rounding="half_up">
          <Number value="2"/>
          <Number value="3"/>
        </arithmetic>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

call:
Llama a un método Java estático para realizar un cálculo complejo.

La expresión `call` declara:

- **type**

  El tipo de datos del valor devuelto (consulte “Tipos de datos soportados” en la página 41) y
- **arguments (opcional)**

  Una lista de valores para pasar como argumentos.

El método Java debe estar en una clase que esté en la vía de acceso de clases en el momento de la validación del conjunto de reglas. El primer argumento en el método debe ser un objeto `Session` y los argumentos restantes deben coincidir con los especificados en el conjunto de reglas.

aviso: Debe asegurarse de que cualquier código Java invocado por una expresión `call` *no* intenta modificar ningún valor en los atributos de objeto de regla.

En general, los conjuntos de reglas CER utilizan tipos de datos inmutables, pero es posible utilizar sus propias clases Java mutables como tipos de datos; si es así, es su responsabilidad asegurarse de que ningún código invocado hace que se modifique el valor de un tipo de datos Java personalizado, porque si esto sucediera podría significar que los cálculos previamente realizados serían ahora "incorrectos".

```
package curam.creole.example;

import curam.creole.execution.RuleObject;
import curam.creole.execution.session.Session;

public class Statics {

  /**
   * Calcula el color favorito de una persona.
   *
   * Este cálculo es demasiado complejo para las reglas y por lo tanto se ha
   * codificado en java.
   *
   * @param session
   *          La sesión de regla
   * @param person
   *          la persona
   * @return el color favorito calculado de la persona especificada
   */
  public static String calculateFavoriteColor(
      final Session session, final RuleObject person) {

    // Tenga en cuenta que la recuperación del valor de atributo debe
    // convertirse al tipo correcto
    final String name =
        (String) person.getAttributeValue("name").getValue();
    final Number age =
        (Number) person.getAttributeValue("age").getValue();

    final String ageString = age.toString();
    // Calcular el color favorito del persona de acuerdo
    // con los dígitos en la edad y el nombre
    if (ageString.contains("5") || ageString.contains("7")) {
      return "Blue";
    } else if (name.contains("z")) {
      return "Purple";
    } else {
      return "Green";
    }

  }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_call"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="age">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="name">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>
```

```
    <Attribute name="favoriteColor">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <!-- Llamar a un método Java estático
             para efectuar el cálculo -->
        <call class="curam.creole.example.Statics"
          method="calculateFavoriteColor">
          <type>
            <javaclass name="String"/>
          </type>
          <arguments>
            <!-- Pasar esta persona
                 como argumento del
                 método estático -->
            <this/>
          </arguments>
        </call>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

PRECAUCIÓN:
Desde Cúram V6, CER y el Gestor de dependencias soportan el recálculo automático de valores calculados por CER si cambian sus dependencias.

Si cambia la implementación de un método estático, CER y el Gestor de dependencias *no* sabrán recalcular automáticamente los valores de atributo que se han calculado utilizando la versión anterior del método estático.

Una vez que se ha utilizado un método estático en un entorno de producción para los valores de atributo almacenados, en lugar de cambiar la implementación deberá crear un método estático nuevo (con la nueva implementación necesaria) y cambiar los conjuntos de reglas para utilizar el método estático nuevo. Cuando publique los cambios de conjunto de reglas para que apunten al nuevo método estático, CER y el Gestor de dependencias recalcularán automáticamente todas las instancias de los valores de atributo afectados.

choose:
Elige un valor basándose en una condición que se cumple.

La expresión `choose` contiene:

- **type**

  un especificador de tipo de datos (consulte “Tipos de datos soportados” en la página 41) que declara el tipo de valor que se elegirá;
- **test (opcional)**

  una expresión que especifica el valor que se debe probar para la `condición` en cada expresión `when` a su vez. Si no se especifica ninguna expresión `test`, la `condición` en cada expresión `when` se prueba a su vez para comprobar si devuelve el valor *true*;
- **when (1 o más)**

  cada uno contiene una `condición` que se debe probar y un `valor` que se debe devolver si la condición pasa y
- **otherwise**

una expresión que contiene un `valor` que se debe devolver (de modo que independientemente de cualquier cosa, se elige siempre un valor).

Las condiciones se evalúan en orden descendente de las expresiones `when`, deteniéndose en la primera condición para pasar la prueba. Las condiciones subsiguientes no se evalúan.

La expresión `choose` refleja las sentencias `if / else if /.../ else` típicas de la mayoría de los lenguajes de programación. Se puede utilizar de forma efectiva para implementar una tabla de decisiones.

Puede tener en cuenta la posibilidad de ordenar las condiciones de manera que las que tienen más probabilidades de ser satisfactorias estén cerca de la parte superior de la lista (para evitar cálculos innecesarios).

aviso: Para condiciones simples (por ejemplo las que prueban la igualdad en un solo valor), normalmente puede reordenar las condiciones sin que ello afecte el comportamiento del conjunto de reglas.

Sin embargo, para condiciones más complejas (y, por tanto, en general), debe considerar detenidamente si la reordenación de las condiciones introducirá cambios de comportamiento no deseados.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_choose"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="age">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="ageCategory">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <choose>
          <!-- No hay ninguna <test> explícita, por lo tanto esta
               sentencia <choose> probará cada condición para
               ver si es verdadera (TRUE). -->
          <type>
            <javaclass name="String"/>
          </type>

          <!-- Tenga en cuenta que el orden de estas condiciones es
               importante; si tuviéramos que intercambiar las posiciones de las
               pruebas "Newborn" e "Infant", todos los niños
               de menos de 5 años (incluidos los que tienen menos de 1)
               se identificarán como bebés (Infants); ningún niño se
               identificará como recién nacido (Newborns).  -->
          <when>
            <condition>
              <compare comparison="&lt;">
                <reference attribute="age"/>
                <Number value="1"/>
```

```
            </compare>
          </condition>
          <value>
            <String value="Newborn"/>
          </value>
        </when>
        <when>
          <condition>
            <compare comparison="&lt;">
              <reference attribute="age"/>
              <Number value="5"/>
            </compare>
          </condition>
          <value>
            <String value="Infant"/>
          </value>
        </when>
        <when>
          <condition>
            <compare comparison="&lt;">
              <reference attribute="age"/>
              <Number value="18"/>
            </compare>
          </condition>
          <value>
            <String value="Hijo/a"/>
          </value>
        </when>
        <otherwise>
          <value>
            <String value="Adult"/>
          </value>
        </otherwise>
      </choose>
    </derivation>
  </Attribute>

  <Attribute name="numberOfSpouses">
    <type>
      <javaclass name="Number"/>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

  <Attribute name="maritalStatus">
    <type>
      <javaclass name="String"/>
    </type>
    <derivation>
      <choose>
        <type>
          <javaclass name="String"/>
        </type>
        <!-- Probar el número de cónyuges -->
        <test>
          <reference attribute="numberOfSpouses"/>
        </test>
        <!-- Tenga en cuenta que el orden de las pruebas "0" y "1" no
importa -
             por lo tanto es aconsejable que los ordene en función de si
la mayoría de
             instancias de persona que se están probando tiene 0 o 1 cónyuges.
     -->
        <when>
          <condition>
```

```
            <Number value="0"/>
          </condition>
          <value>
            <String value="Unmarried"/>
          </value>
        </when>
        <when>
          <condition>
            <Number value="1"/>
          </condition>
          <value>
            <String value="Casado/a - cónyuge único"/>
          </value>
        </when>
        <otherwise>
          <value>
            <String value="Casado/a - varios cónyuges"/>
          </value>
        </otherwise>
      </choose>

    </derivation>
  </Attribute>

</Class>

</RuleSet>
```

Code:
Un valor constante de literal que representa un código de una tabla de códigos de aplicación.

La expresión `Code` especifica un nombre de tabla de códigos y toma un solo argumento que especifica el valor del código necesario de la tabla.

Nota: Debe especificar el valor de serie del código; no se pueden utilizar constantes generadas de tabla de códigos, porque CER es un lenguaje totalmente dinámico y no puede depender de construcciones de tiempo de compilación.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_Code"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <!-- Representación booleana de género -->
    <Attribute name="isMale">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- Representación de código de género -->
    <Attribute name="gender">
      <type>
        <codetableentry table="Gender"/>
      </type>
      <derivation>
        <Code table="Gender">
          <choose>
            <type>
              <javaclass name="String"/>
```

```
          </type>
          <when>
            <condition>
              <reference attribute="isMale"/>
            </condition>
            <value>
              <!-- utilizar el código "MALE" de la tabla de códigos -->
              <String value="MALE"/>
            </value>
          </when>
          <otherwise>
            <value>
              <!-- utilizar el código "FEMALE" de la tabla de códigos -->
              <String value="FEMALE"/>
            </value>
          </otherwise>
        </choose>
      </Code>
    </derivation>
  </Attribute>

</Class>

</RuleSet>
```

combineSuccessionSets:
Consulte la guía Inside Cúram Eligibility and Entitlement Using Cúram Express Rules.

compare:
Compara un valor del lado izquierdo con un valor del lado derecho, de acuerdo con la comparación proporcionada.

Las comparaciones soportadas son:

- `<`
  el lado izquierdo "es menor que" el lado derecho;
- `<=`
  el lado izquierdo "es menor que o igual a" el lado derecho;
- `>`
  el lado izquierdo "es mayor que" el lado derecho; y
- `>=`
  el lado izquierdo "es mayor que o igual a" el lado derecho.

Los valores de lado izquierdo y lado derecho pueden ser de cualquier tipo de objeto comparable, incluyendo (pero sin limitarse a):

- `Number`;
- `String` y
- `curam.util.type.Date`.

Nota: Todas las instancias de `Number` se convierten al formato numérico propio de CER (respaldado por `java.math.BigDecimal`) antes de la comparación.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_compare"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="CompareExampleRuleClass">
```

```
    <!-- 3 >= 2 - TRUE-->
    <Attribute name="compareTwoNumbers">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <compare comparison=">=">
          <Number value="3"/>
          <Number value="2"/>
        </compare>
      </derivation>
    </Attribute>

    <!-- Año Nuevo anterior a Navidad - TRUE -->
    <Attribute name="compareTwoDates">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <compare comparison="&lt;">
          <Date value="2007-01-01"/>
          <Date value="2007-12-25"/>
        </compare>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

concat:
Crea un mensaje traducible (consulte "Soporte de localización" en la página 8) concatenando una lista de valores.

Las series `concat` encadena sus valores sin texto o espacios adicionales; si necesita un formato más complejo o texto traducible, considere la posibilidad de utilizar "`ResourceMessage`" en la página 226 en su lugar.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_concat"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="firstName">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="surname">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="dateOfBirth">
      <type>
        <javaclass name="curam.util.type.Date"/>
```

```
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- Un identificador para una persona, incluidos
     nombre, apellido y fecha de nacimiento, por ej.
     Juan Herrero (03 de octubre de 1970).

     Nombre y apellido son series simples,
     pero la fecha de nacimiento será traducible
     de acuerdo con la configuración regional del usuario.
     -->
    <Attribute name="personIdentifier">
      <type>
        <javaclass name="curam.creole.value.Message"/>
      </type>
      <derivation>
        <concat>
          <fixedlist>
            <listof>
              <!-- Observe que utilizamos Object, porque tenemos una
                   mezcla de elementos String y Date
                   en la lista. -->
              <javaclass name="Object"/>
            </listof>
            <members>
              <reference attribute="firstName"/>
              <!-- space separator between names -->
              <String value=" "/>
              <reference attribute="surname"/>
              <String value=" ("/>
              <reference attribute="dateOfBirth"/>
              <String value=")"/>
            </members>
          </fixedlist>
        </concat>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

create:
Obtiene una instancia nueva de una clase de regla en la memoria de la sesión. Los valores de inicialización necesarios para el objeto de regla debe especificarse como elementos hijo de la expresión `create`.

Desde Cúram V6, se puede utilizar `create` para crear una instancia nueva de una clase de regla de un conjunto de reglas *diferente*, estableciendo el valor del atributo XML `ruleset` opcional.

Nota: Los objetos de regla creados utilizando `create` no se pueden recuperar durante la ejecución de reglas, porque si se hiciera se violaría el principio de orden de CER.

Desde Cúram V6, hay una opción de sintaxis al pasar valores a un objeto de regla creado:

- **bloque de inicialización**

CER continúa soportando un bloque de atributos definidos en un elemento `Initialization`. Esta sintaxis puede ser útil para los atributos que se *deben* establecer siempre y no tienen ninguna implementación predeterminada y

- **elementos specify**

  Desde Cúram V6, CER también soporta atributos arbitrarios cuyo valor se anula temporalmente por el uso de un elemento `specify` que nombra el atributo a establecer y que contiene el valor a utilizar. Esta sintaxis puede ser útil para los atributos que sólo se establecen a veces y/o tienen una implementación predeterminada.

Desde Cúram V6, los objetos de regla creados se "agrupan" en la sesión. Esta agrupación permite que las solicitudes idénticas creen un objeto de regla que deberá ser atendido por un solo objeto de regla, que puede conservar el uso de memoria y también evitar que tengan lugar cálculos idénticos, lo que hace que se reduzca la carga de la CPU. Dos solicitudes para crear un objeto de regla se consideran idénticas si solicitan la misma clase de regla y los valores de todos los atributos inicializados y especificados son iguales.

En el ejemplo siguiente, si el número de teléfono del trabajo de una persona es idéntico al número de teléfono de casa de la persona, se utilizará un solo objeto de regla para cada número y, por consiguiente, el valor derivado para `isOutOfThisArea` sólo se calculará una vez. De lo contrario, si los números de teléfono de trabajo y de casa son diferentes, se crearán dos objetos de regla diferentes.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_create"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">

  <Class name="Person">

    <!-- Detalles de número de teléfono reunidos como pruebas -->
    <Attribute name="homePhoneAreaCode">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="homePhoneNumber">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="workPhoneAreaCode">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="workPhoneNumber">
```

```
    <type>
      <javaclass name="Number"/>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

  <!-- Crear objetos de regla PhoneNumber
       y ponerlos en una lista -->
  <Attribute name="phoneNumbers">
    <type>
      <javaclass name="List">
        <ruleclass name="PhoneNumber"/>
      </javaclass>
    </type>
    <derivation>
      <fixedlist>
        <listof>
          <ruleclass name="PhoneNumber"/>
        </listof>

        <members>

          <!-- Número de teléfono para los detalles de casa. -->
          <create ruleclass="PhoneNumber">
            <!-- El valor para PhoneNumber.owner -->
            <this/>
            <!-- El valor para PhoneNumber.number -->
            <reference attribute="homePhoneNumber"/>
            <specify attribute="areaCode">
              <!-- El valor para PhoneNumber.areaCode -->
              <reference attribute="homePhoneAreaCode"/>
            </specify>
          </create>

          <!-- Número de teléfono para los detalles de trabajo.

               Si el número de teléfono de trabajo de una persona es idéntico al
               número de teléfono de casa de la persona (es decir, el prefijo local y el
               número son los mismos), esta expresión <create>
               devolverá el mismo objeto de regla que el objeto de regla
               devuelto por la expresión <create> anterior.  Si los
               números de teléfono no son idénticos, se devolverán dos
               objetos de regla diferentes.-->
          <create ruleclass="PhoneNumber">
            <this/>
            <reference attribute="workPhoneNumber"/>
            <specify attribute="areaCode">
              <reference attribute="workPhoneAreaCode"/>
            </specify>
          </create>

        </members>
      </fixedlist>
    </derivation>
  </Attribute>

</Class>

<Class name="PhoneNumber">

  <Initialization>
    <!-- Los valores para estos atributos los debe pasar, en orden,
         cualquier expresión <create>. -->
    <Attribute name="owner">
      <type>
```

```
        <ruleclass name="Person"/>
      </type>
    </Attribute>
    <Attribute name="number">
      <type>
        <javaclass name="Number"/>
      </type>
    </Attribute>
  </Initialization>


  <!-- El valor de este atributo se puede pasar mediante un elemento <specify>
       en una expresión <create>, que alterará temporalmente
       aquí la derivación predeterminada. -->
  <Attribute name="areaCode">
    <type>
      <javaclass name="Number"/>
    </type>
    <derivation>
      <!-- Implementación predeterminada, utilizada si la expresión <create>
           no <especifica> un valor para este atributo. -->
      <Number value="123"/>
    </derivation>
  </Attribute>

  <!-- Para un objeto de regla agrupado, este valor derivado sólo se
       calculará una vez.

       Por ejemplo, si el número de teléfono de trabajo de una persona es idéntico
       al número de teléfono de casa de la persona, se utilizará el mismo objeto
       de regla para los números de teléfono de casa y del trabajo
       y el valor "isOutOfThisArea" para este objeto de regla
       individual se calculará una sola vez.
    -->
  <Attribute name="isOutOfThisArea">
    <type>
      <javaclass name="Boolean"/>
    </type>
    <derivation>
      <not>
        <equals>
          <reference attribute="areaCode"/>
          <!-- El prefijo local para el área de la agencia -->
          <Number value="123"/>
        </equals>
      </not>
    </derivation>
  </Attribute>
 </Class>
</RuleSet>
```

current:
Hace referencia a un elemento de una lista que se está procesando.

La expresión `current` sólo puede aparecer en una expresión que proceso elementos de una lista, por ejemplo:

- `listitemexpression` en una expresión "`filter`" en la página 196 o "`dynamiclist`" en la página 189 o
- `sortorder` en una expresión "`sort`" en la página 230.

Para mayor claridad, puede asignar un alias a la expresión `current`, que debe coincidir con el alias en la expresión `list` a la que se hace referencia. Los alias son necesarios si hay más que expresiones `current` en el ámbito en el mismo cálculo.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_listitem"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Household">

    <Attribute name="members">
      <type>
        <javaclass name="List">
          <ruleclass name="Person"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="adults">
      <type>
        <javaclass name="List">
          <ruleclass name="Person"/>
        </javaclass>
      </type>
      <derivation>
        <filter>
          <list>
            <reference attribute="members"/>
          </list>
          <listitemexpression>
            <!-- La referencia utiliza current para hacer referencia
                   a un elemento de la lista de
                   objetos de regla de persona. -->
            <reference attribute="isAdult">
              <current/>
            </reference>
          </listitemexpression>
        </filter>
      </derivation>
    </Attribute>

  </Class>

  <Class name="Person">

    <Attribute name="children">
      <type>
        <javaclass name="List">
          <ruleclass name="Person"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="age">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="isAdult">
      <type>
```

```
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <compare comparison=">=">
          <reference attribute="age"/>
          <Number value="18"/>
        </compare>

      </derivation>
    </Attribute>

    <!-- Los hijos de esta persona que
         aún no son adultos. -->
    <Attribute name="dependentChildren">
      <type>
        <javaclass name="List">
          <ruleclass name="Person"/>
        </javaclass>
      </type>
      <derivation>
        <filter>
          <!-- Utilizar un alias para evitar confusión (para personas
               que leen el conjunto de reglas) entre la persona padre
               y la persona hijo.  -->
          <list alias="child">
            <reference attribute="children"/>
          </list>
          <listitemexpression>
            <not>
              <reference attribute="isAdult">
                <!-- El alias de current debe coincidir
                   con el de la lista. -->
                <current alias="child"/>
              </reference>
            </not>
          </listitemexpression>
        </filter>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

Date:
Un valor constante literal de fecha, del tipo `curam.util.type.Date`.

El valor de `Date` se especifica en el formato *aaaa-mm-dd*.

Nota: Intencionadamente no hay ninguna función en CER para obtener la fecha actual: una función de este tipo sería volátil porque hoy devolvería un valor y mañana un valor diferente.

Las funciones volátiles están prohibidas en CER, porque si el resultado de una función puede cambiar, puede significar que los cálculos realizados previamente son ahora "incorrectos".

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_Date"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="DateExampleRuleClass">

    <Attribute name="nullDate">
```

```
      <type>
        <javaclass name="curam.util.type.Date"/>
      </type>
      <derivation>
        <!-- Una fecha nula -->
        <null/>
      </derivation>
    </Attribute>

    <Attribute name="dateOfBirth">
      <type>
        <javaclass name="curam.util.type.Date"/>
      </type>
      <derivation>
        <!-- La fecha 3 de octubre de 1970 -->
        <Date value="1970-10-03"/>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

dynamiclist:
Crea una nueva lista evaluando una expresión en cada elemento de una lista existente.

La nueva lista tendrá una entrada correspondiente a cada entrada de la lista existente, con el orden conservado.

Una expresión `dynamiclist` especifica:

- **list**

  La lista existente y
- **listitemexpression**

  La expresión para evaluar cada elemento de la lista existente.

Se puede utilizar un `dynamiclist` cuando el número de elementos de la lista deseada no se conoce como tiempo de diseño (es decir, puede diferir de una ejecución a otra, en función del valor de otros atributos). Si el número de elementos es fijo (es decir, se conoce el tiempo de diseño), tenga en cuenta la posibilidad de utilizar "`fixedlist`" en la página 199 en su lugar.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_dynamiclist"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="age">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="isDisabled">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
```

```
      <specified/>
    </derivation>
  </Attribute>

  <Attribute name="totalIncome">
    <type>
      <javaclass name="Number"/>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

  <Attribute name="pets">
    <type>
      <javaclass name="List">
        <ruleclass name="Pet"/>
      </javaclass>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

</Class>

<Class name="Pet">
  <Initialization>
    <Attribute name="name">
      <type>
        <javaclass name="String"/>
      </type>
    </Attribute>
  </Initialization>

</Class>

<Class name="Household">

  <Attribute name="members">
    <type>
      <javaclass name="List">
        <ruleclass name="Person"/>
      </javaclass>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

  <Attribute name="containsDisabledPerson">
    <type>
      <javaclass name="Boolean"/>
    </type>
    <derivation>
      <any>
        <!-- obtiene una lista de booleanos, correspondiente
             al atributo isDisabled en cada
             miembro de persona de esta unidad familiar   -->
        <dynamiclist>
          <list>
            <reference attribute="members"/>
          </list>
          <listitemexpression>
            <reference attribute="isDisabled">
              <current/>
            </reference>
```

```
        </listitemexpression>
      </dynamiclist>
    </any>
  </derivation>
</Attribute>

<Attribute name="totalIncomeOfAdultMembers">
  <type>
    <javaclass name="Number"/>
  </type>
  <derivation>
    <sum>
      <dynamiclist>
        <list>
          <!-- filtrar los miembros hacia abajo hasta
               sólo los adultos -->
          <filter>
            <list>
              <reference attribute="members"/>
            </list>
            <listitemexpression>
              <compare comparison=">=">
                <reference attribute="age">
                  <current/>
                </reference>
                <Number value="18"/>
              </compare>
            </listitemexpression>
          </filter>
        </list>
        <listitemexpression>
          <reference attribute="totalIncome">
            <current/>
          </reference>
        </listitemexpression>
      </dynamiclist>
    </sum>
  </derivation>
</Attribute>

<Attribute name="memberAges">
  <type>
    <javaclass name="List">
      <javaclass name="Number"/>
    </javaclass>
  </type>
  <derivation>
    <dynamiclist>
      <list>
        <reference attribute="members"/>
      </list>
      <listitemexpression>
        <reference attribute="age">
          <current/>
        </reference>
      </listitemexpression>
    </dynamiclist>
  </derivation>
</Attribute>

<Attribute name="youngestAge">
  <type>
    <javaclass name="Number"/>
  </type>
  <derivation>
    <min>
      <reference attribute="memberAges"/>
```

```
        </min>
      </derivation>
    </Attribute>

    <!-- obtener todas las mascotas de la unidad familiar,
         uniendo la lista de
         mascotas de cada persona -->
    <Attribute name="allPets">
      <type>
        <javaclass name="List">
          <ruleclass name="Pet"/>
        </javaclass>
      </type>
      <derivation>
        <joinlists>
          <!-- una lista de mascotas, una
               lista para cada miembro de
               unidad familiar -->
          <dynamiclist>
            <list>
              <reference attribute="members"/>
            </list>
            <listitemexpression>
              <reference attribute="pets">
                <current/>
              </reference>
            </listitemexpression>
          </dynamiclist>

        </joinlists>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

defaultDescription:
Proporciona una implementación predeterminada del atributo `description` que todas las clases de regla heredan de la clase de regla raíz. Consulte "Tipos de datos soportados" en la página 41.

Cada clase de regla debe alterar temporalmente el atributo `description` de la clase de regla raíz para proporcionar una descripción más significativa. Si no se proporciona (o hereda) ninguna alteración temporal, se emitirá un aviso cuando se valide el conjunto de reglas.

Importante: La expresión `defaultDescription` *sólo* debe utilizarla la clase de regla raíz; el usuario no debe utilizarla en sus propias clases de regla.

```
<Class name="RootRuleClass" abstract="true">
    <Attribute name="description">
      <type>
        <javaclass name="curam.creole.value.Message"/>
      </type>
      <derivation>
        <!-- Para uso SÓLO en RootRuleClass -->
        <defaultDescription/>
      </derivation>
    </Attribute>
  </Class>
```

equals:
Determina si dos objetos (un objeto de lado izquierdo y un objeto del lado derecho) son iguales.

Los valores de Number se convierten al propio formato numérico de CER (respaldado por java.math.BigDecimal) antes de la comparación; las diferencias en los ceros iniciales o de cola se ignoran.

Los valores null se comparan sin riesgo; si el lado izquierdo y el lado derecho son null, la expresión equals devuelve true; si sólo uno de los valores del lado izquierdo y lado derecho es null, la expresión equals devuelve false.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_equals"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="EqualsExampleRuleClass">

    <!-- VERDADERO -->
    <Attribute name="identicalStrings">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <equals>
          <String value="A String"/>
          <String value="A String"/>
        </equals>
      </derivation>
    </Attribute>

    <!-- FALSO -->
    <Attribute name="differentStrings">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <equals>
          <String value="A String"/>
          <String value="A different String"/>
        </equals>
      </derivation>
    </Attribute>

    <!-- VERDADERO -->
    <Attribute name="identicalNumbers">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <equals>
          <!-- Estos números son iguales,
            ignorando las diferencias
            triviales en los ceros
            iniciales/de cola -->
          <Number value="123"/>
          <Number value="000123.000"/>
        </equals>
      </derivation>
    </Attribute>

    <!-- FALSO -->
    <Attribute name="differentTypes">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <equals>
          <!-- Estos objetos son de
```

```
            tipos diferentes, por lo tanto
            no son iguales aunque
            "parezcan" iguales.-->
          <String value="123"/>
          <Number value="123"/>
        </equals>
      </derivation>
    </Attribute>

    <!-- FALSO -->
    <Attribute name="oneNull">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <equals>
          <null/>
          <Number value="456"/>
        </equals>
      </derivation>
    </Attribute>

    <!-- VERDADERO -->
    <Attribute name="twoNulls">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <equals>
          <null/>
          <null/>
        </equals>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

existencetimeline:
Crea una línea de tiempo de un tipo especificado a partir de un par de fechas de inicio y finalización inclusive, cada una de las cuales es opcional.

Consulte “Construcción de líneas de tiempo” en la página 60.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_existencetimeline"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">

  <Class name="Person">

    <Attribute name="dateOfBirth">
      <type>
        <javaclass name="curam.util.type.Date"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>


    <!-- será nulo si el persona aún está activa -->
    <Attribute name="dateOfDeath">
      <type>
        <javaclass name="curam.util.type.Date"/>
```

```
  </type>
  <derivation>
    <specified/>
  </derivation>
</Attribute>

<!-- Crea una línea de tiempo que es falsa antes de que nazca
     la persona, verdadera mientras la persona está viva y falsa después de que
     muera la persona.   Si la persona no tiene ninguna fecha de defunción registrada,
     no habrá ningún intervalo falso ("false") final. -->
<Attribute name="isAliveTimeline">
  <type>
    <javaclass name="curam.creole.value.Timeline">
      <javaclass name="Boolean"/>
    </javaclass>
  </type>
  <derivation>
    <existencetimeline>
      <intervaltype>
        <javaclass name="Boolean"/>
      </intervaltype>
      <intervalfromdate>
        <reference attribute="dateOfBirth"/>
      </intervalfromdate>
      <intervaltodate>
        <reference attribute="dateOfDeath"/>
      </intervaltodate>
      <preExistenceValue>
        <false/>
      </preExistenceValue>
      <existenceValue>
        <true/>
      </existenceValue>
      <postExistenceValue>
        <false/>
      </postExistenceValue>
    </existencetimeline>

  </derivation>
</Attribute>

<!-- Crea una línea de tiempo que es antes del nacimiento ("Before Birth") antes de que
     nazca la persona, durante el tiempo de vida ("During Lifetime") mientras la persona está v
     después de la muerte ("After Death") después de que muera la persona.   Si la persona no t
     ninguna fecha de defunción registrada, no habrá ningún intervalo de después de la muerte (
     Death") final. -->
<Attribute name="lifeStatus">
  <type>
    <javaclass name="curam.creole.value.Timeline">
      <javaclass name="String"/>
    </javaclass>
  </type>
  <derivation>
    <existencetimeline>
      <intervaltype>
        <javaclass name="String"/>
      </intervaltype>
      <intervalfromdate>
        <reference attribute="dateOfBirth"/>
      </intervalfromdate>
      <intervaltodate>
        <reference attribute="dateOfDeath"/>
      </intervaltodate>
      <preExistenceValue>
        <String value="Before Birth"/>
      </preExistenceValue>
      <existenceValue>
```

```
          <String value="During Lifetime"/>
        </existenceValue>
        <postExistenceValue>
          <String value="After Death"/>
        </postExistenceValue>
      </existencetimeline>

    </derivation>
  </Attribute>



</Class>
</RuleSet>
```

false:
El valor constante booleano “false”.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_false"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="FalseExampleRuleClass">

    <Attribute name="isCuramExpertRulesFantastic">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <not>
          <false/>
        </not>
      </derivation>
    </Attribute>

    <Attribute name="didCookbookWinPulitzerPrize">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <false/>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

filter:
Crea una nueva lista que contiene todos los elementos de una lista existente que cumplen la condición de filtro.

La expresión `filter` contiene:

- **list**

  una lista existente para filtrar y
- **listitemexpression**

  la prueba a aplicar a cada elemento de la lista.

Normalmente, `listitemexpression` contiene uno o varios cálculos aplicados al elemento “current” en la página 186 en la lista.

El orden relativo de los elementos de lista en el resultado filtrado conservará el orden relativo de los elementos de lista en la lista original. Si ninguno de los elementos de la lista cumplen la condición de filtro, se devuelve una lista vacía.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_filter"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="age">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- El cónyuge de esta persona o
      nulo si no está casada -->
    <Attribute name="spouse">
      <type>
        <ruleclass name="Person"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- Los hijos de esta persona -->
    <Attribute name="children">
      <type>
        <javaclass name="List">
          <ruleclass name="Person"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>



  </Class>



  <Class name="Household">

    <!-- Todas las personas de la unidad familiar -->
    <Attribute name="members">
      <type>
        <javaclass name="List">
          <ruleclass name="Person"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- Todos los adultos de la unidad familiar -->
    <Attribute name="adultMembers">
      <type>
        <javaclass name="List">
```

```
      <ruleclass name="Person"/>
    </javaclass>
  </type>
  <derivation>
    <filter>
      <list>
        <reference attribute="members"/>
      </list>
      <listitemexpression>
        <compare comparison=">=">
          <reference attribute="age">
            <current/>
          </reference>
          <Number value="18"/>
        </compare>
      </listitemexpression>
    </filter>
  </derivation>
</Attribute>

<!-- Todos los progenitores solos de la unidad familiar -->
<Attribute name="loneParents">
  <type>
    <javaclass name="List">
      <ruleclass name="Person"/>
    </javaclass>
  </type>
  <derivation>
    <filter>
      <list>
        <reference attribute="members"/>
      </list>
      <listitemexpression>
        <all>
          <fixedlist>
            <listof>
              <javaclass name="Boolean"/>
            </listof>
            <members>

              <!-- Sin cónyuge -->
              <equals>
                <reference attribute="spouse">
                  <current/>
                </reference>
                <null/>
              </equals>
              <!-- Como mínimo un hijo -->
              <not>
                <property name="isEmpty">
                  <object>
                    <reference attribute="children">
                      <current/>
                    </reference>
                  </object>
                </property>
              </not>
            </members>
          </fixedlist>
        </all>
      </listitemexpression>
    </filter>
  </derivation>
</Attribute>
```

```
  </Class>

</RuleSet>
```

fixedlist:
Crea una nueva lista de elementos conocidos en el momento del diseño de conjunto de reglas.

La expresión `fixedlist` especifica:

- **listof**

  El tipo de elemento en la lista devuelta (consulte "Tipos de datos soportados" en la página 41) y
- **members**

  Los elementos de la lista.

La lista creada contendrá sus miembros en el orden listado en el conjunto de reglas.

Consejo: El elemento `members` puede contener 0, 1 o muchos elementos hijo.

Sin embargo, si `fixedlist` está contenido en una operación de proceso de lista pero sólo especifica 0 o 1 miembros de lista, el validador de conjunto de reglas CER emitirá un aviso, indicando que la lista puede ser innecesaria.

Si necesita crear una lista donde el número de elementos de la lista no se conoce en el momento del diseño, considere la posibilidad de utilizar "`dynamiclist`" en la página 189 en su lugar.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_fixedlist"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <!-- Las mascotas de propiedad de esta persona -->
    <Attribute name="pets">
      <type>
        <javaclass name="List">
          <ruleclass name="Pet"/>
        </javaclass>
      </type>
      <derivation>

        <!-- Una lista fija de mascotas -->
        <fixedlist>
          <listof>
            <ruleclass name="Pet"/>
          </listof>
          <members>
            <!-- Cada persona tiene exactamente dos mascotas,
                 Skippy y Lassie -->
            <create ruleclass="Pet">
              <String value="Skippy"/>
              <String value="Kangaroo"/>
            </create>
            <create ruleclass="Pet">
              <String value="Lassie"/>
              <String value="Dog"/>
            </create>
          </members>
```

```
        </fixedlist>
      </derivation>
    </Attribute>


    <Attribute name="isEntitledToBenefits">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <all>
          <!-- Una lista fija de condiciones booleanas -->
          <fixedlist>
            <listof>
              <javaclass name="Boolean"/>
            </listof>
            <members>
              <!-- Debe ser adulto -->
              <compare comparison=">=">
                <reference attribute="age"/>
                <Number value="18"/>
              </compare>
              <!-- Debe ser residente en el estado -->
              <reference attribute="isResidentInTheState"/>
              <!-- Debe tener ingresos por debajo del umbral para prestaciones
-->
              <compare comparison="&lt;">
                <reference attribute="totalIncome"/>
                <Number value="100"/>
              </compare>
            </members>
          </fixedlist>

        </all>

      </derivation>
    </Attribute>

    <Attribute name="totalIncome">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <!-- Una suma inútil de un elemento -
             el validador de conjunto de reglas CER avisará de que esta
             fixedlist puede ser innecesaria. -->
        <sum>
          <fixedlist>
            <listof>
              <javaclass name="Number"/>
            </listof>
            <members>
              <!-- Sumar sólo los ingresos salariales -->
              <reference attribute="earnedIncome"/>
            </members>
          </fixedlist>

        </sum>

      </derivation>
    </Attribute>

    <Attribute name="earnedIncome">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
```

```
        <specified/>
      </derivation>
    </Attribute>


    <Attribute name="isResidentInTheState">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="age">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

  </Class>

  <Class name="Pet">

    <Initialization>

      <Attribute name="name">
        <type>
          <javaclass name="String"/>
        </type>
      </Attribute>

      <Attribute name="species">
        <type>
          <javaclass name="String"/>
        </type>
      </Attribute>

    </Initialization>

  </Class>

</RuleSet>
```

FrequencyPattern:
Un valor constante FrequencyPattern literal, de tipo `curam.util.type.FrequencyPattern`.

El valor de `FrequencyPattern` se especifica como un número de 9 dígitos. Consulte el JavaDoc para `curam.util.type.FrequencyPattern` para conocer el significado de la serie de dígitos.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_FrequencyPattern"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="FrequencyPatternExampleRuleClass">

    <Attribute name="nullFrequencyPattern">
      <type>
        <javaclass name="curam.util.type.FrequencyPattern"/>
      </type>
```

```
      <derivation>
        <!-- Un FrequencyPattern nulo -->
        <null/>
      </derivation>
    </Attribute>

    <Attribute name="weeklyOnMondays">
      <type>
        <javaclass name="curam.util.type.FrequencyPattern"/>
      </type>
      <derivation>
        <!-- La serie de Patrón de frecuencia para
             "Weekly on Mondays" -->
        <FrequencyPattern value="100100100"/>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

Intervalo:
Crea un intervalo (consulte “Manejo de datos que cambian a lo largo del tiempo” en la página 51) de un tipo determinado, con un valor válido a partir de una fecha especificada.

Esta expresión suele utilizarse como parte de la construcción de una “Timeline” en la página 237.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_Interval"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">

  <Class name="CreateInterval">

    <Attribute name="aNumberTimeline">
      <type>
        <javaclass name="curam.creole.value.Timeline">
          <javaclass name="Number"/>
        </javaclass>
      </type>
      <derivation>
        <Timeline>
          <intervaltype>
            <javaclass name="Number"/>
          </intervaltype>
          <initialvalue>
            <Number value="0"/>
          </initialvalue>
          <!-- Another interval-->
          <intervals>
            <fixedlist>
              <listof>
                <javaclass name="curam.creole.value.Interval">
                  <javaclass name="Number"/>
                </javaclass>
              </listof>
              <members>
                <!-- Crea un intervalo del tipo especificado.
                     Normalmente se utiliza como entrada en una <Línea de tiempo>. -->
                <Interval>
                  <intervaltype>
                    <javaclass name="Number"/>
                  </intervaltype>
```

```
                <start>
                  <Date value="2001-01-01"/>
                </start>
                <value>
                  <Number value="10000"/>
                </value>
              </Interval>
            </members>
          </fixedlist>

        </intervals>
      </Timeline>

    </derivation>
  </Attribute>


 </Class>
</RuleSet>
```

intervalvalue:
Reinicia una expresión que devuelve una línea de tiempo (consulte "Manejo de datos que cambian a lo largo del tiempo" en la página 51) y permite que una expresión que contiene opere en los valores individuales en la línea de tiempo. Esta expresión hace de forma efectiva que una expresión externa no "sepa" que está operando en una línea de tiempo.

Esta expresión sólo puede utilizarse cuando está anidada en una expresión "timelineoperation" en la página 239. Para obtener más ejemplos de uso y descripciones adicionales de intervalvalue, consulte "timelineoperation" en la página 239.

joinlists:
Crea una nueva lista uniendo algunas listas existentes.

La expresión joinlists toma un solo argumento que debe ser una lista de listas.

El orden de los elementos de la nueva lista es idéntico al orden de la lista de origen. Las listas se unen en el orden en que se proporcionan.

Si las listas que se unen pueden contener elementos duplicados, considere la posibilidad de envolver la expresión joinlists en una expresión "removeduplicates" en la página 224.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_joinlists"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="pets">
      <type>
        <javaclass name="List">
          <ruleclass name="Pet"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

  </Class>
```

```
  <Class name="Pet">
    <Initialization>
      <Attribute name="name">
        <type>
          <javaclass name="String"/>
        </type>
      </Attribute>
    </Initialization>

  </Class>

  <Class name="Household">

    <Attribute name="members">
      <type>
        <javaclass name="List">
          <ruleclass name="Person"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- obtener todas las mascotas de la unidad familiar,
         uniendo la lista de
         mascotas de cada persona -->
    <Attribute name="allPets">
      <type>
        <javaclass name="List">
          <ruleclass name="Pet"/>
        </javaclass>
      </type>
      <derivation>
        <joinlists>
          <!-- una lista de mascotas, una
               lista para cada miembro de
               unidad familiar -->
          <dynamiclist>
            <list>
              <reference attribute="members"/>
            </list>
            <listitemexpression>
              <reference attribute="pets">
                <current/>
              </reference>
            </listitemexpression>
          </dynamiclist>

        </joinlists>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

legislationChange:
Consulte la guía Inside Cúram Eligibility and Entitlement Using Cúram Express Rules.

max:
Determina el valor más grande de una lista (o null si la lista está vacía).

La lista puede contener cualquier tipo de objeto comparable, incluyendo (pero sin limitarse a):

- `Number`;
- `String` y
- `curam.util.type.Date`.

Nota: Todas las instancias de `Number` se convierten al formato numérico propio de CER (respaldado por `java.math.BigDecimal`) antes de la comparación.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_max"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="MaxExampleRuleClass">

    <!-- Elegiremos "Cherry" como el valor de serie "más grande" -->
    <Attribute name="alphabeticallyLastFruit">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <max>
          <reference attribute="fruits"/>
        </max>
      </derivation>
    </Attribute>

    <Attribute name="fruits">
      <type>
        <javaclass name="List">
          <javaclass name="String"/>
        </javaclass>
      </type>
      <derivation>
        <fixedlist>
          <listof>
            <javaclass name="String"/>
          </listof>
          <members>
            <String value="Apple"/>
            <String value="Banana"/>
            <String value="Cherry"/>
          </members>
        </fixedlist>
      </derivation>
    </Attribute>

    <!-- Determina el número de manchas del perro con más manchas -->
    <Attribute name="largestNumberOfSpots">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <max>
          <dynamiclist>
            <list>
              <reference attribute="dalmatians"/>
            </list>
            <listitemexpression>
              <reference attribute="numberOfSpots">
                <current/>
              </reference>
            </listitemexpression>
          </dynamiclist>
```

```
      </max>
    </derivation>
  </Attribute>

  <Attribute name="dalmatians">
    <type>
      <javaclass name="List">
        <ruleclass name="Dalmation"/>
      </javaclass>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

</Class>

<Class name="Dalmation">

  <Attribute name="numberOfSpots">
    <type>
      <javaclass name="Number"/>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

</Class>

</RuleSet>
```

min:
Determina el valor más pequeño de una lista (o `null` si la lista está vacía).

La lista puede contener cualquier tipo de objeto comparable, incluyendo (pero sin limitarse a):

- `Number`;
- `String` y
- `curam.util.type.Date`.

Nota: Todas las instancias de `Number` se convierten al formato numérico propio de CER (respaldado por `java.math.BigDecimal`) antes de la comparación.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_min"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="MinExampleRuleClass">

    <!-- Elegiremos Año Nuevo como el valor de la fecha "más temprana" -->
    <Attribute name="eariestDate">
      <type>
        <javaclass name="curam.util.type.Date"/>
      </type>
      <derivation>
        <min>
          <reference attribute="publicHolidays"/>
        </min>
      </derivation>
    </Attribute>

    <Attribute name="publicHolidays">
      <type>
```

```
        <javaclass name="List">
          <javaclass name="curam.util.type.Date"/>
        </javaclass>
      </type>
      <derivation>
        <fixedlist>
          <listof>
            <javaclass name="curam.util.type.Date"/>
          </listof>
          <members>
            <Date value="2007-01-01"/>
            <Date value="2007-12-25"/>
          </members>
        </fixedlist>
      </derivation>
    </Attribute>

    <!-- Determina el número de rayas de la cebra menos
 rayada-->
    <Attribute name="smallestNumberOfStripes">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <min>
          <dynamiclist>
            <list>
              <reference attribute="zebras"/>
            </list>
            <listitemexpression>
              <reference attribute="numberOfStripes">
                <current/>
              </reference>
            </listitemexpression>
          </dynamiclist>
        </min>
      </derivation>
    </Attribute>

    <Attribute name="zebras">
      <type>
        <javaclass name="List">
          <ruleclass name="Zebra"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

  </Class>

  <Class name="Zebra">

    <Attribute name="numberOfStripes">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

not:
Niega un valor booleano.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_not"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="isLivingInUSA">

      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <!-- Tenga en cuenta que este 'no dentro de no' es algo forzado.
 -->
        <not>
          <reference attribute="isLivingOutsideUSA"/>
        </not>
      </derivation>

    </Attribute>

    <Attribute name="isLivingOutsideUSA">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <not>
          <equals>
            <reference attribute="country"/>
            <String value="USA"/>
          </equals>
        </not>
      </derivation>
    </Attribute>

    <!-- El país donde reside esta persona. -->
    <Attribute name="country">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

null:
Una constante valor `null` (nulo).

El establecimiento de un valor en `null` puede ser útil para indicar que no se aplica ningún valor.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_null"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Pet">
    <Initialization>
      <Attribute name="name">
```

```
      <type>
        <javaclass name="String"/>
      </type>
    </Attribute>
  </Initialization>

</Class>

<Class name="Person">

  <!-- La mascota favorita de la persona o
       nulo si la persona no es propietaria de ninguna mascota. -->
  <Attribute name="favoritePet">
    <type>
      <ruleclass name="Pet"/>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

  <!-- El nombre de la mascota
       favorita de la persona o nulo si
       la persona no es propietaria de ninguna mascota.

       Tenemos que probar que favoritePet
       es nulo antes de realizar el
       cálculo (simple).-->
  <Attribute name="favoritePetsName">
    <type>
      <javaclass name="String"/>
    </type>
    <derivation>
      <choose>
        <type>
          <javaclass name="String"/>
        </type>
        <when>
          <!-- si esta persona no tiene ninguna
               mascota favorita, calcular el
               nombre de la mascota favorita como nulo. -->
          <condition>
            <equals>
              <reference attribute="favoritePet"/>
              <null/>
            </equals>
          </condition>
          <value>
            <null/>
          </value>
        </when>
        <otherwise>
          <value>
            <!-- obtener el nombre de la mascota favorita -->
            <reference attribute="name">
              <reference attribute="favoritePet"/>
            </reference>
          </value>
        </otherwise>
      </choose>

    </derivation>
  </Attribute>

</Class>

</RuleSet>
```

Número:
Un valor constante de número de literal.

Un número en CER es un valor decimal de longitud arbitraria, especificado utilizando un punto (“.”) como separador decimal y sin ningún separador de millares.

Los cálculos empresariales de CER pueden implicar normalmente valores de porcentaje, por ejemplo "Deducir el 10% de los ingresos de la personas". Para obtener ayuda con la codificación de estas reglas, CER permite especificar un número como porcentaje, simplemente poniendo como sufijo el número con %. Por ejemplo, los números *12,345%* y *0,12345* se comportarán de manera idéntica en los cálculos (pero la versión de porcentaje se visualizará en formato de porcentaje).

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_Number"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="NumberExampleRuleClass">

    <Attribute name="aPositiveInteger">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <!-- Un entero positivo -->
        <Number value="1"/>
      </derivation>
    </Attribute>

    <Attribute name="aNegativeInteger">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <!-- Un entero negativo -->
        <Number value="-2"/>
      </derivation>
    </Attribute>

    <Attribute name="aDecimalNumber">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <!-- Un número decimal.

             Los números son arbitrariamente largos/precisos, utilizan "." para
             el separador decimal y no tienen separador
             de miles.
        -->
        <Number value="-12345.6789"/>
      </derivation>
    </Attribute>

    <Attribute name="aPercentage">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <!-- Un porcentaje
            (12,345% equivale al número 0,12345) -->
        <Number value="12.345%"/>
      </derivation>
```

```
    </Attribute>

  </Class>

</RuleSet>
```

periodlength:
Calcula la cantidad de unidades de tiempo entre dos fechas.

Se debe especificar una de las siguientes unidades de tiempo:

- *días*;
- *semanas*;
- *meses* y
- *años*.

La expresión `periodlength` también debe especificar si la fecha de finalización del periodo es *inclusive* o *exclusive* o la fecha de finalización (el periodo es siempre incluyendo la fecha de inicio).

El cálculo de la duración del periodo siempre se redondea al entero más cercan inferior.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_periodlength"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="PeriodLengthExampleClass">

    <!-- NB 1970 no era un año bisiesto -->
    <Attribute name="firstDayOfJanuary1970">
      <type>
        <javaclass name="curam.util.type.Date"/>
      </type>
      <derivation>
        <Date value="1970-01-01"/>
      </derivation>
    </Attribute>

    <Attribute name="lastDayOfDecember1970">
      <type>
        <javaclass name="curam.util.type.Date"/>
      </type>
      <derivation>
        <Date value="1970-12-31"/>
      </derivation>
    </Attribute>

    <Attribute name="firstDayOfJanuary1971">
      <type>
        <javaclass name="curam.util.type.Date"/>
      </type>
      <derivation>
        <Date value="1971-01-01"/>
      </derivation>
    </Attribute>

    <Attribute name="sameDay_LengthInDays">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <!-- se inicia y finaliza el mismo día = 1 día -->
```

```
    <periodlength endDateInclusion="inclusive" unit="days">
      <reference attribute="firstDayOfJanuary1970"/>
      <reference attribute="firstDayOfJanuary1970"/>
    </periodlength>
  </derivation>
</Attribute>

<Attribute name="sameDay_LengthInWeeks">
  <type>
    <javaclass name="Number"/>
  </type>
  <derivation>
    <!-- se inicia y finaliza el mismo día = 0 semanas-->
    <periodlength endDateInclusion="exclusive" unit="weeks">
      <reference attribute="firstDayOfJanuary1970"/>
      <reference attribute="firstDayOfJanuary1970"/>
    </periodlength>
  </derivation>
</Attribute>

<Attribute name="januaryToDecember_LengthInDays">
  <type>
    <javaclass name="Number"/>
  </type>
  <derivation>
    <!-- 365 days -->
    <periodlength endDateInclusion="inclusive" unit="days">
      <reference attribute="firstDayOfJanuary1970"/>
      <reference attribute="lastDayOfDecember1970"/>
    </periodlength>
  </derivation>
</Attribute>

<Attribute name="januaryToDecember_LengthInYearsExclusive">
  <type>
    <javaclass name="Number"/>
  </type>
  <derivation>
    <!-- 0 años (casi 1 año, pero sólo 1 día escasamente) -->
    <periodlength endDateInclusion="exclusive" unit="years">
      <reference attribute="firstDayOfJanuary1970"/>
      <reference attribute="lastDayOfDecember1970"/>
    </periodlength>
  </derivation>
</Attribute>

<Attribute name="januaryToDecember_LengthInYearsInclusive">
  <type>
    <javaclass name="Number"/>
  </type>
  <derivation>
    <!-- 1 year (exactly) -->
    <periodlength endDateInclusion="inclusive" unit="years">
      <reference attribute="firstDayOfJanuary1970"/>
      <reference attribute="lastDayOfDecember1970"/>
    </periodlength>
  </derivation>
</Attribute>


<Attribute name="januaryToJanuary_LengthInYearsExclusive">
  <type>
    <javaclass name="Number"/>
  </type>
  <derivation>
    <!-- 1 year (exactly) -->
    <periodlength endDateInclusion="exclusive" unit="years">
```

```
          <reference attribute="firstDayOfJanuary1970"/>
          <reference attribute="firstDayOfJanuary1971"/>
        </periodlength>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

property:
Obtiene una propiedad de un objeto Java.

La expresión `property` especifica el nombre del método Java a llamar y:

- **object**
  El objeto Java en el que se debe operar y
- **arguments**
  Opcionalmente una lista de argumentos a pasar al método Java.

La expresión `property` permite a CER aprovechar la potencia de las clases Java sin tener que replicar un subconjunto arbitrario de métodos como expresiones CER. Por ejemplo, `java.util.List` contiene un método `size` y por lo tanto CER no contiene ninguna expresión explícita para calcular el número de elementos de una lista.

Sin embargo, para satisfacer el principio de inmutabilidad de CER, sólo se puede llamar a los métodos Java que no modifican el "valor" de ningún objeto. CER sólo permite llamar a un método "property" si el método se incluye en la "lista segura" de métodos para la clase del objeto (o uno de las clases o interfaces ancestro).

Se considera que un método es seguro si se ha marcado explícitamente como tal en la lista segura. Si no está presente en la lista seguridad, el validador de conjunto de reglas CER emitirá un error.

Consejo: El establecimiento explícito de seguridad como *false* no es necesario, pero puede incluirse para completar la documentación, como sucede con las listas seguras que se incluyen con CER.

La lista segura para una clase es un archivo de propiedades en el mismo paquete que la clase, denominado *<nombre_clase>_CREOLE.properties*.

CER incluye listas seguras para las siguientes clases e interfaces Java:

- `curam.creole.value.Timeline`;
- `java.lang.Object`;
- `java.lang.Number`; and
- `java.util.List`.

Lista segura para los métodos `curam.creole.value.Timeline`.

```
# Lista segura para curam.creole.value.Timeline

# seguro
valueOn.safe=true
```

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_property"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
```

```
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">

  <Class name="Person">
    <Attribute name="isMinor">
      <type>
        <javaclass name="curam.creole.value.Timeline">
          <ruleclass name="Boolean"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- Si esta persona es un menor. -->
    <Attribute name="isAChild">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <property name="valueOn">
          <object>
            <reference attribute="isMinor"/>
          </object>
          <arguments>
            <Date value="2000-01-01"/>
          </arguments>
        </property>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

Lista segura para los métodos `java.lang.Object`.

```
# Lista segura para java.lang.Object

# seguro
toString.safe=true

# forzar que se evalúe la igualdad utilizando <equals>
equals.safe=false

# no se exponen, aunque son "seguros"
hashCode.safe=false
getClass.safe=false
```

Lista segura para los métodos `java.lang.Number`.

```
# Lista segura para java.lang.Number

byteValue.safe=true
doubleValue.safe=true
floatValue.safe=true
intValue.safe=true
longValue.safe=true
shortValue.safe=true
```

Lista segura para los métodos `java.util.List`.

```
# Lista segura para java.util.List

contains.safe=true
containsAll.safe=true
```

```
get.safe=true

indexOf.safe=true
isEmpty.safe=true
lastIndexOf.safe=true
size.safe=true
subList.safe=true


# no expuesto
hashCode.safe=false
listIterator.safe=false
iterator.safe=false
toArray.safe=false

# mutadores - no seguro
add.safe=false
addAll.safe=false
clear.safe=false
remove.safe=false
removeAll.safe=false
retainAll.safe=false
```

Para obtener una descripción de algunas de las propiedades útiles en la interfaz Java `List`, consulte "Operaciones de lista útiles" en la página 252.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_property"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">

  <Class name="Person">
    <Attribute name="children">
      <type>
        <javaclass name="List">
          <ruleclass name="Person"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- Indica si esta persona tiene hijos.

         Prueba la propiedad isEmpty de lista. -->
    <Attribute name="hasChildren">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <not>
          <property name="isEmpty">
            <object>
              <reference attribute="children"/>
            </object>
          </property>
        </not>

      </derivation>
    </Attribute>

    <!-- Todos los hijos de esta persona, excluyendo el primer hijo.

         Utiliza la propiedad subList de lista, pasando:
         - (inclusive) del elemento en la posición "1" (que indica el
```

```
segundo
          miembro de la lista; las listas en Java están basadas en cero)
        - (exclusive) al elemento en la posición "tamaño de la lista" (que indica
          la posición posterior al último elemento de la lista)
   -->
   <Attribute name="secondAndSubsequentChildren">
     <type>
       <javaclass name="List">
         <ruleclass name="Person"/>
       </javaclass>
     </type>
     <derivation>
       <property name="subList">
         <object>
           <reference attribute="children"/>
         </object>
         <arguments>
           <!-- El número se debe convertir a un entero
                (como lo requiere List.subList). -->
           <property name="intValue">
             <object>
               <Number value="1"/>
             </object>
           </property>
           <property name="size">
             <object>
               <reference attribute="children"/>
             </object>
           </property>

         </arguments>
       </property>

     </derivation>
   </Attribute>

 </Class>

</RuleSet>
```

PRECAUCIÓN:
Desde Cúram V6, CER y el Gestor de dependencias soportan el almacenamiento de los valores de atributo calculados en la base de datos, junto con el recálculo automático de valores de atributo si las dependencias cambian.

Si cambia la implementación de un método de propiedad, CER y el Gestor de dependencias *no* sabrán recalcular automáticamente los valores de atributo que se han calculado utilizando la versión antigua del método de propiedad.

Una vez que se ha utilizado un método de propiedad en un entorno de producción para valores de atributo almacenados, en lugar de cambiar la implementación deberá crear un nuevo método de propiedad (con la nueva implementación necesaria) y cambiar los conjuntos de reglas para utilizar el nuevo método de propiedad. Cuando publique los cambios de conjunto de regla para que apunten al nuevo método de propiedad, CER recalculará automáticamente todas las instancias de los valores de atributo afectados.

rate:
Consulte la guía Inside Cúram Eligibility and Entitlement Using Cúram Express Rules.

readall:

Recupera todas las instancias de objeto de regla externa de una clase de regla (es decir, los creados por el código de cliente). Las instancias de objeto de regla interna (es decir, los creados a partir de reglas) *no* se recuperan.

Consulte "Objetos de regla externa e interna" en la página 33 para obtener más detalles sobre la creación de objetos de regla.

Desde Cúram V6, se puede utilizar `readall` para recuperar instancias de una clase de regla de un conjunto de reglas *diferente*, estableciendo el valor del atributo XML `ruleset` opcional.

Desde Cúram V6, la expresión `readall` soporta un elemento `match` opcional que hace que la expresión `readall` sólo recupere objetos de regla cuyo valor para un atributo determinado coincide con el de los criterios de búsqueda.

Importante: Antes de Curam V6, una manera de recuperar objetos de regla que coincidan con un criterio consistía en envolver un `readall` en una expresión "`filter`" en la página 196.

Sin embargo, en el caso de las sesiones de CER que utilizan un DatabaseDataStorage (consulte "Sesiones de CER" en la página 30), en general será más efectivo utilizar la sintaxis `readall / match` presentada en Cúram V6. La nueva sintaxis funcionará mejor:

- cuando el atributo que contiene la expresión `readall` se calcule primero y
- cuando CER y el Gestor de dependencia identifiquen que el atributo que contiene la expresión `readall` está anticuado y es necesario volver a calcularlo (consulte "Gestor de dependencias" en la página 80).

En situaciones en las que los objetos de regla deben coincidir en más de un criterio, debe utilizar la sintaxis `readall / match` para que coincida en el atributo más selectivo y, a continuación, envolver los resultados en un "`filter`" en la página 196 para filtrar adicionalmente por los demás criterios.

Consejo: Si sólo espera que haya una única instancia de la clase de regla (quizá después del filtrado o la coincidencia), tenga en cuenta la posibilidad de envolver la expresión en una expresión "`singleitem`" en la página 228.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_readall"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="socialSecurityNumber">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!--
    Recuperar la única reclamación que se tendrá que haber utilizado para
    iniciar la sesión
    -->

    <Attribute name="claim">
      <type>
```

```
    <ruleclass name="Claim"/>
  </type>
  <derivation>
    <singleitem onEmpty="error" onMultiple="error">
      <readall ruleclass="Claim"/>
    </singleitem>
  </derivation>
</Attribute>


<!--
Recuperar los objetos de regla de prestación para esta persona (creados a partir del
código de cliente, probablemente consultando el almacenamiento externo).

Esta implementación utiliza un <readall> con un <match> anidado para
recuperar sólo los objetos de regla coincidentes y (en función del
almacenamiento de datos) será más efectiva que la
implementación de "benefitsFilterReadall" que se muestra a continuación.
-->

<Attribute name="benefitsReadallMatch">
  <type>
    <javaclass name="List">
      <ruleclass name="Benefit"/>
    </javaclass>
  </type>
  <derivation>
    <readall ruleclass="Benefit">
      <match retrievedattribute="socialSecurityNumber">
        <reference attribute="socialSecurityNumber"/>
      </match>
    </readall>
  </derivation>
</Attribute>


<!--
Recupera los mismos objetos de regla que para "benefitsReadallMatch"
mostrado más arriba, pero (en función del almacenamiento de datos) puede no ser tan efectivo.
-->
<Attribute name="benefitsFilterReadall">
  <type>
    <javaclass name="List">
      <ruleclass name="Benefit"/>
    </javaclass>
  </type>
  <derivation>
    <filter>
      <list>
        <!-- recuperar todos los objetos de regla de prestación de
             almacenamiento externo -->
        <readall ruleclass="Benefit"/>
      </list>
      <listitemexpression>
        <equals>
          <!-- coincidir con los números de la seguridad social en
            el objeto de regla de persona y el objeto de
            regla de prestación -->
          <reference attribute="socialSecurityNumber">
            <current/>
          </reference>
          <reference attribute="socialSecurityNumber"/>
        </equals>
      </listitemexpression>
    </filter>
  </derivation>
</Attribute>
```

```
<!--
Recupera las prestaciones de la persona de tipo "IncomeAssistance",
utilizando un <match> para recuperar todas las prestaciones de la persona y, a continuación,
un <filter> para extraer sólo las prestaciones de "Ayuda a los ingresos" de
las prestaciones para dicha persona.

Esta implementación puede ser adecuada cuando
socialSecurityNumber sea el atributo más selectivo para una
prestación en el almacenamiento de datos (es decir, hay muchos objetos de regla
de prestación, pero cada valor socialSecurityNumber está presente en
relativamente pocos objetos de regla de prestación).
-->
<Attribute name="incomeAssistanceBenefitsMatchSSNFilterType">
  <type>
    <javaclass name="List">
      <ruleclass name="Benefit"/>
    </javaclass>
  </type>
  <derivation>
    <filter>
      <list>
        <!-- recuperar todos los objetos de regla de prestación para la persona
-->
        <readall ruleclass="Benefit">
          <match retrievedattribute="socialSecurityNumber">
            <reference attribute="socialSecurityNumber"/>
          </match>
        </readall>
      </list>
      <listitemexpression>
        <equals>
          <!-- filtrar los objetos de regla de prestación para la persona
               sólo hasta los de tipo "Ayuda a los ingresos"
            -->
          <reference attribute="type">
            <current/>
          </reference>
          <Code table="BenefitType">
            <!-- El valor para Ayuda a los ingresos -->
            <String value="BT1"/>
          </Code>
        </equals>
      </listitemexpression>
    </filter>
  </derivation>
</Attribute>


<!--
Recupera las prestaciones de la persona de tipo "IncomeAssistance",
utilizando una coincidencia (<match>) para recuperar todas las prestaciones de "Ayuda a los ir
y, a continuación, un filtro (<filter>) para extraer sólo las prestaciones de "Ayuda a
los ingresos" para esta persona.

Esta implementación puede ser adecuada cuando el tipo sea el
atributo más selectivo para una prestación en el almacenamiento de datos (es decir,
hay pocos objetos de regla de prestación para cada tipo).
-->
<Attribute name="incomeAssistanceBenefitsMatchTypeFilterSSN">
  <type>
    <javaclass name="List">
      <ruleclass name="Benefit"/>
    </javaclass>
  </type>
  <derivation>
    <filter>
```

```
      <list>
        <!-- recuperar todos los objetos de regla de prestación de tipo "Ayuda a
             los ingresos" -->
        <readall ruleclass="Benefit">
          <match retrievedattribute="type">
            <Code table="BenefitType">
              <!-- El valor para Ayuda a los ingresos -->
              <String value="BT1"/>
            </Code>
          </match>
        </readall>
      </list>
      <listitemexpression>
        <equals>
          <!-- filtrar los objetos de regla de prestación de tipo "Ayuda a
               los ingresos" hasta los correspondientes a esta persona solamente
            -->
          <reference attribute="socialSecurityNumber">
            <current/>
          </reference>
          <reference attribute="socialSecurityNumber"/>
        </equals>
      </listitemexpression>
    </filter>
  </derivation>
</Attribute>

<!--
Recupera los objetos de regla para prestaciones cuyo importe es mayor
que 100.

Dado que "mayor que" no es un predicado de coincidencia exacta,
se debe utilizar un <filter> (<match> sólo se puede utilizar para un
criterio de coincidencia exacta).
-->
<Attribute name="highPaymentBenefits">
  <type>
    <javaclass name="List">
      <ruleclass name="Benefit"/>
    </javaclass>
  </type>
  <derivation>
    <filter>
      <list>
        <!-- recuperar todos los objetos de regla de prestación para la persona
          -->
        <readall ruleclass="Benefit">
          <match retrievedattribute="socialSecurityNumber">
            <reference attribute="socialSecurityNumber"/>
          </match>
        </readall>
      </list>
      <listitemexpression>
        <!-- filtrar los objetos de regla de prestación para la persona
             sólo para aquellos de un importe superior a 100 -->
        <compare comparison=">">
          <reference attribute="amount">
            <current/>
          </reference>
          <Number value="100"/>
        </compare>
      </listitemexpression>
    </filter>
  </derivation>
</Attribute>
```

```
  </Class>

  <Class name="Benefit">

    <Attribute name="socialSecurityNumber">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="type">
      <type>
        <codetableentry table="BenefitType"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="amount">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

  </Class>

  <!--
  Este conjunto de reglas espera que el código que crea la sesión también
  crea una sola instancia de programa de arranque ("bootstrap") de esta
  clase de regla de reclamación.
  -->
  <Class name="Claim">
    <Initialization>
      <Attribute name="claimIdentifier">
        <type>
          <javaclass name="String"/>
        </type>
      </Attribute>
      <Attribute name="claimDate">
        <type>
          <javaclass name="curam.util.type.Date"/>
        </type>
      </Attribute>
    </Initialization>
  </Class>

</RuleSet>
```

reference:
Recupera el valor de un atributo de un objeto de regla.

La expresión `reference` puede contener opcionalmente una expresión hijo que determina el objeto de regla a partir de la cual se puede obtener el atributo; si se omite, se utilizará el objeto de regla que contiene `reference`.

La expresión `reference` es la clave para crear reglas que son reutilizables y comprensibles. Puede utilizar `reference` en un atributo con nombre en lugar de

cualquier expresión. El validador de conjunto de reglas CER emitirá un error si el tipo de atributo de referencia no coincide con el tipo requerido por la expresión.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_reference"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="name">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="age">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- Una simple referencia a otro
         atributo en esta clase de regla -->
    <Attribute name="ageNextYear">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <arithmetic operation="+">
          <!-- Esta <reference> no tiene elementos hijo,
               de modo que se toma el objeto de regla para que sea "este
 objeto de regla"-->
          <reference attribute="age"/>
          <Number value="1"/>
        </arithmetic>
      </derivation>
    </Attribute>

    <Attribute name="favoritePet">
      <type>
        <ruleclass name="Pet"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

  </Class>

  <Class name="Pet">

    <Attribute name="name">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>
```

```
  <Attribute name="species">
    <type>
      <javaclass name="String"/>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

</Class>

<Class name="Household">

  <!-- Todas las personas de la unidad familiar -->
  <Attribute name="members">
    <type>
      <javaclass name="List">
        <ruleclass name="Person"/>
      </javaclass>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

  <!-- Se designa a una persona especial
       como el "cabeza" de familia -->
  <Attribute name="headOfHousehold">
    <type>
      <ruleclass name="Person"/>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

  <!-- Una referencia a un atributo en un
       objeto de regla diferente:

  name OF favoritePet OF headOfHousehold

  En un lenguaje de programación, esto se puede
  escribir al revés utilizando una notación de
  "puntos" de desreferencia como la siguiente:

  headOfHousehold.favoritePet.name

  -->
  <Attribute name="nameOfHeadOfHouseholdsFavoritePet">
    <type>
      <javaclass name="String"/>
    </type>
    <derivation>

      <!-- El nombre... -->
      <reference attribute="name">

        <!-- ...de la mascota favorita... -->
        <reference attribute="favoritePet">

          <!-- ...del cabeza de familia. -->
          <!-- La referencia más interna debe hacer referencia a
                 un atributo en este objeto de regla.  -->
          <reference attribute="headOfHousehold"/>
        </reference>
      </reference>
    </derivation>
```

```
    </Attribute>


    <!-- Identifica a los propietarios de perros de la unidad familiar
         comprobando qué personas tienen una mascota
         favorita que sea un perro. -->
    <Attribute name="dogOwners">
      <type>
        <javaclass name="List">
          <ruleclass name="Person"/>
        </javaclass>
      </type>
      <derivation>
        <filter>
          <list>
            <!-- referencia simple a los miembros
                 de la unidad familiar -->
            <reference attribute="members"/>
          </list>
          <listitemexpression>
            <equals>
              <String value="Dog"/>
              <!-- Una referencia a un atributo en un elemento
                   en la lista. -->

              <reference attribute="species">
                <reference attribute="favoritePet">
                  <current/>
                </reference>
              </reference>

            </equals>

          </listitemexpression>
        </filter>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

removeduplicates:
Crea una lista nueva eliminando los elementos duplicados de una lista existente.

Si algún elemento de la lista original aparece más de una vez, sólo se conserva la primera instancia. De lo contrario, se mantiene el orden de los elementos.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_removeduplicates"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <!-- La lista de relaciones donde
         esta persona es la "fromPerson". -->
    <Attribute name="relationships">
      <type>
        <javaclass name="List">
          <ruleclass name="Relationship"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>
```

```
  <!-- Las personas que están relacionadas con esta persona.

       Cualquier familiar aparece sólo una vez en esta lista,
       aunque una persona pueda estar
       relacionada con otra de más de una manera, por ejemplo
       mi abuelo también puede ser mi tutor legal.-->
  <Attribute name="uniqueRelatives">
    <type>
      <javaclass name="List">
        <ruleclass name="Person"/>
      </javaclass>
    </type>
    <derivation>
      <removeduplicates>
        <reference attribute="allRelatives"/>
      </removeduplicates>
    </derivation>
  </Attribute>

  <Attribute name="allRelatives">
    <type>
      <javaclass name="List">
        <ruleclass name="Person"/>
      </javaclass>
    </type>
    <derivation>
      <!-- obtener los familiares de esta persona formando una
           lista de la "toPerson" en el otro extremo de cada
           relación. -->
      <dynamiclist>
        <list>
          <reference attribute="relationships"/>
        </list>
        <listitemexpression>
          <reference attribute="toPerson">
            <current/>
          </reference>
        </listitemexpression>
      </dynamiclist>
    </derivation>
  </Attribute>

</Class>

<!-- Una relación de una persona con otra. -->
<Class name="Relationship">

  <Attribute name="fromPerson">
    <type>
      <ruleclass name="Person"/>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

  <Attribute name="relationshipType">
    <type>
      <javaclass name="String"/>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>
```

```
    <Attribute name="toPerson">
      <type>
        <ruleclass name="Person"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

ResourceMessage:
Crea un mensaje traducible (consulte "Soporte de localización" en la página 8) a partir de un recurso de propiedad.

Opcionalmente, la propiedad puede especificar marcadores para argumentos formateados. El soporte y la sintaxis para el formato se describe en el JavaDoc para `MessageFormat`.

aviso: Como se menciona en el JavaDoc, si necesita producir una comilla simple o apóstrofo (’), debe especificar *dos* comillas simples en el texto de propiedad (’’).

Si necesita producir XML o HTML y no necesita formato de símbolos complejo o la posibilidad de cambiar texto de mensaje sin cambiar reglas, tenga en cuenta la posibilidad de utilizar "`XmlMessage`" en la página 246 en su lugar.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_ResourceMessage"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="gender">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="isMarried">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="firstName">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="surname">
      <type>
        <javaclass name="String"/>
```

```
  </type>
  <derivation>
    <specified/>
  </derivation>
</Attribute>

<Attribute name="income">
  <type>
    <javaclass name="Number"/>
  </type>
  <derivation>
    <specified/>
  </derivation>
</Attribute>

<!-- Devuelve un saludo que puede
    salir en la configuración regional del usuario -->
<Attribute name="simpleGreetingMessage">
  <type>
    <javaclass name="curam.creole.value.Message"/>
  </type>
  <derivation>
    <ResourceMessage key="simpleGreeting"
      resourceBundle="curam.creole.example.Messages"/>
  </derivation>
</Attribute>

<!-- Devuelve un saludo que contiene
     el título y el apellido de la persona.
     El saludo y el título se pueden localizar,
     el apellido no (es idéntico
     en todas las configuraciones regionales). -->
<Attribute name="parameterizedGreetingMessage">
  <type>
    <javaclass name="curam.creole.value.Message"/>
  </type>
  <derivation>
    <!-- pasar argumentos a
         los marcadores de mensaje -->
    <ResourceMessage key="parameterizedGreeting"
      resourceBundle="curam.creole.example.Messages">
      <!-- Título -->
      <choose>
        <type>
          <javaclass name="curam.creole.value.Message"/>
        </type>
        <when>
          <condition>
            <equals>
              <reference attribute="gender"/>
              <String value="Male"/>
            </equals>
          </condition>
          <value>
            <ResourceMessage key="title.male"
              resourceBundle="curam.creole.example.Messages"/>
          </value>
        </when>
        <when>
          <condition>
            <reference attribute="isMarried"/>
          </condition>
          <value>
            <ResourceMessage key="title.female.married"
              resourceBundle="curam.creole.example.Messages"/>
          </value>
        </when>
```

```
            <otherwise>
              <value>
                <ResourceMessage key="title.female.single"
                  resourceBundle="curam.creole.example.Messages"/>
              </value>
            </otherwise>
          </choose>

          <!-- Apellido -->
          <reference attribute="surname"/>

        </ResourceMessage>
      </derivation>
    </Attribute>

    <!-- Formatea un número de 2 decimales,
         con coma decimal y separador de
         miles en la configuración regional del usuario -->
    <Attribute name="incomeStatementMessage">
      <type>
        <javaclass name="curam.creole.value.Message"/>
      </type>
      <derivation>
        <ResourceMessage key="incomeStatement"
          resourceBundle="curam.creole.example.Messages">
          <reference attribute="income"/>
        </ResourceMessage>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

Propiedades de ejemplo, Inglés.

```
# file curam/creole/example/Messages_en.properties

simpleGreeting=Hello
parameterizedGreeting=Hello, {0} {1}
title.male=Mr.
title.female.single=Miss
title.female.married=Mrs.
incomeStatement=Income: USD{0,number,#0.00}
```

Propiedades de ejemplo, Francés.

```
# file curam/creole/example/Messages_fr.properties

simpleGreeting=Bonjour
parameterizedGreeting=Bonjour, {0} {1}
title.male=M.
title.female.single=Mlle.
title.female.married=Mme.
incomeStatement=Revenue: EUR{0,number,#0.00}
```

singleitem:
Recupera un único elemento de una lista.

La expresión `singleitem` puede ser útil cuando se espera que una lista sólo contenga un único elemento, por ejemplo al filtrar una lista por criterios que sólo debe seleccionar un solo elemento de la lista.

La expresión `singleitem` especifica:

- **onEmpty**

El comportamiento cuando se encuentra que la lista está vacía:

– **error**

  Se produce un error de tiempo de ejecución (utilice esta opción si no se espera que la lista esté vacía) o

– **returnNull**

  Se devuelve el valor *null*.

- **onMultiple**

  El comportamiento cuando se encuentra que la lista contiene más de un elemento:

  – **error**

    Se produce un error de tiempo de ejecución (utilice esta opción si no se espera que la lista contenga más de un elemento);

  – **returnNull**

    Se devuelve el valor *null*;

  – **returnFirst**

    Se devuelve el primer elemento de la lista o

  – **returnLast**

    Se devuelve el último elemento de la lista.

Para recuperar un elemento de una posición específica en una lista, consulte `get` en "Operaciones de lista útiles" en la página 252.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_singleitem"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="dateOfBirth">
      <type>
        <javaclass name="curam.util.type.Date"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <Attribute name="children">
      <type>
        <javaclass name="List">
          <ruleclass name="Person"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- El primer hijo nacido de esta persona -->
    <Attribute name="firstBornChild">
      <type>
        <ruleclass name="Person"/>
      </type>
      <derivation>
        <!-- obtener el primer hijo, si hay alguno
             - si no hay hijos, devolver nulo -->
        <singleitem onEmpty="returnNull" onMultiple="returnFirst">
          <!-- clasificar los hijos por orden de fecha de nacimiento -->
```

```
          <sort>
            <list alias="child">
              <reference attribute="children"/>
            </list>
            <sortorder>
              <sortitem direction="ascending">
                <reference attribute="dateOfBirth">
                  <current alias="child"/>
                </reference>
              </sortitem>
            </sortorder>
          </sort>

        </singleitem>
      </derivation>

    </Attribute>

    <!-- Recuperar el único registro de información de la unidad familiar
         del almacenamiento externo - siempre debe haber
         exactamente uno - cualquier otro valor es un error.  -->
    <Attribute name="householdInformation">
      <type>
        <ruleclass name="HouseholdInformation"/>
      </type>
      <derivation>
        <singleitem onEmpty="error" onMultiple="error">
          <readall ruleclass="HouseholdInformation"/>
        </singleitem>
      </derivation>
    </Attribute>

  </Class>

  <Class name="HouseholdInformation">

    <Attribute name="householdContainsDisabledPerson">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

sort:
Crea una lista nueva clasificando los miembros de una lista existente de acuerdo con un orden de clasificación especificado.

Una expresión `sort` especifica:

- **list**

  La lista existente que se debe clasificar (que no quedará afectada) y
- **sortorder**

  El orden en el que desea clasificar la lista.

`sortorder` especifica uno o más `sortitem`, cada uno de los cuales especifica el elemento por el que se debe clasificar y si se debe clasificar en orden ascendente o descendente.

Los sortitem se listan poniendo primero el más significativo; cada sortitem sólo se evalúa si dos elementos que se están clasificando son idénticos en relación a los sortitem más significativos.

Dentro de cada sortitem, puede utilizar "current" en la página 186 para hacer referencia al elemento de lista que se está clasificando. Normalmente cada sortitem hará referencia a algún atributo o cálculo en el elemento de lista "current" en la página 186.

Si dos (o más) elementos de la lista son idénticos respecto a todos los sortitem, se devuelven en el mismo orden relativo que la lista de origen.

El comportamiento de la expresión sort es similar al de la cláusula ORDER BY de SQL.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_sort"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Household">

    <Attribute name="members">
      <type>
        <javaclass name="List">
          <ruleclass name="Person"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- Ordena los miembros en orden de edad (más viejo a más joven);
         para miembros que son la misma edad, los miembros se
         ordenan en orden alfabético por nombre. -->
    <Attribute name="sortedMembers">
      <type>
        <javaclass name="List">
          <ruleclass name="Person"/>
        </javaclass>
      </type>
      <derivation>
        <sort>
          <list>
            <reference attribute="members"/>
          </list>
          <sortorder>
            <sortitem direction="descending">
              <!-- La edad de la persona en la lista -->
              <reference attribute="age">
                <current/>
              </reference>
            </sortitem>
            <!-- El nombre de la persona en la lista -->
            <sortitem direction="ascending">
              <reference attribute="firstName">
                <current/>
              </reference>
            </sortitem>
          </sortorder>
        </sort>
      </derivation>
    </Attribute>
```

```
  </Class>

  <Class name="Person">

    <Initialization>
      <Attribute name="firstName">
        <type>
          <javaclass name="String"/>
        </type>
      </Attribute>
      <Attribute name="age">
        <type>
          <javaclass name="Integer"/>
        </type>
      </Attribute>
    </Initialization>

  </Class>

</RuleSet>
```

specified:
Una expresión de marcador para indicar que el valor del atributo se especifica externamente (por ejemplo, por recuperación de una base de datos o llenado del código de prueba), en lugar de que lo calcule el proceso de reglas.

Normalmente, los atributos `especificado` indican información que viene directamente de fuera del sistema y otros atributos utilizan esta información externa para obtener información nueva.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_specified"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <!-- Esta información no se puede calcular u obtener -
         debe especificarse desde un origen externo -->
    <Attribute name="name">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- Esta información no se puede calcular u obtener -
         debe especificarse desde un origen externo -->
    <Attribute name="dateOfBirth">
      <type>
        <javaclass name="curam.util.type.Date"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- Es probable que otros atributos obtengan/calculen más
         información basándose en los atributos "especificados" más arriba -->
  </Class>
</RuleSet>
```

String:
Un valor constante de serie (String) literal.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_String"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="StringExampleRuleClass">

    <Attribute name="emptyString">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <!-- Una serie vacía -->
        <String value=""/>
      </derivation>
    </Attribute>

    <Attribute name="helloWorld">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <!-- La serie "Hello, World!" -->
        <String value="Hello, World!"/>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

sublists:
Calcula todas las sublistas de la lista proporcionada y devuelve estas sublistas como una lista de listas.

Para una lista que contiene *n* elementos, hay 2^n sublistas, incluyendo la lista vacía y la lista original.

El orden de los elementos de lista en cada una de las sublistas será idéntico al orden de la lista original.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_sublists"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Household">

    <Attribute name="members">
      <type>
        <javaclass name="List">
          <ruleclass name="Person"/>
        </javaclass>
      </type>
      <derivation>
        <fixedlist>
          <listof>
            <ruleclass name="Person"/>
          </listof>
          <members>
            <create ruleclass="Person">
              <String value="Madre"/>
            </create>
            <create ruleclass="Person">
              <String value="Padre"/>
            </create>
```

```
            <create ruleclass="Person">
              <String value="Hijo/a"/>
            </create>
          </members>
        </fixedlist>
      </derivation>
    </Attribute>

    <!-- Todas las combinaciones diferentes de los miembros de la unidad familiar
 -->
    <Attribute name="memberCombinations">
      <!-- Tenga en cuenta que el tipo es lista de listas de personas -->
      <type>
        <javaclass name="List">
          <javaclass name="List">
            <ruleclass name="Person"/>
          </javaclass>
        </javaclass>
      </type>
      <derivation>
        <sublists>
          <reference attribute="members"/>
        </sublists>
      </derivation>
    </Attribute>

  </Class>

  <Class name="Person">

    <Initialization>
      <Attribute name="name">
        <type>
          <javaclass name="String"/>
        </type>
      </Attribute>
    </Initialization>

  </Class>

</RuleSet>
```

En este conjunto de reglas de ejemplo, el valor de *memberCombinations* se calcula como una lista de estas 8 listas:

- una lista vacía (sin miembros de unidad familiar);
- Madre;
- Padre;
- Madre y Padre;
- Hijo/a;
- Madre e Hijo/a;
- Padre e Hijo/a y
- Madre, Padre e Hijo/a (la lista completa original).

sum:
Calcula la suma numérica de una lista de valores de número.

Si la lista está vacía, esta expresión devuelve *0*.

La lista de valores de número la suele proporcionar "fixedlist" en la página 199 o "dynamiclist" en la página 189.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_sum"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <Attribute name="netWorth">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <!-- Ejemplo de <sum> operando en una <fixedlist> -->
        <!-- El valor neto de una persona es la suma de su
             efectivo, sus ahorros y sus activos -->
        <sum>
          <fixedlist>
            <listof>
              <javaclass name="Number"/>
            </listof>
            <members>
              <reference attribute="totalCash"/>
              <reference attribute="totalSavings"/>
              <reference attribute="totalAssets"/>
            </members>
          </fixedlist>
        </sum>
      </derivation>
    </Attribute>

    <Attribute name="totalAssets">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <!-- Ejemplo de <sum> operando en una <dynamiclist> -->
        <!-- El valor total de los activos de una persona se obtiene
             sumando el valor de cada activo -->
        <sum>
          <dynamiclist>
            <list>
              <reference attribute="assets"/>
            </list>
            <listitemexpression>
              <reference attribute="value">
                <current/>
              </reference>
            </listitemexpression>
          </dynamiclist>
        </sum>
      </derivation>
    </Attribute>

    <!-- Los activos que son propiedad de esta persona -->
    <Attribute name="assets">
      <type>
        <javaclass name="List">
          <ruleclass name="Asset"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- NB este ejemplo no muestra cómo
         se obtiene el total de efectivo/ahorros -->
```

```
    <Attribute name="totalCash">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>
    <Attribute name="totalSavings">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

  </Class>

  <Class name="Asset">

    <!-- El valor monetario del activo -->
    <Attribute name="value">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

this:
Una referencia al objeto de regla actual, similar a la palabra clave this en Java.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_this"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="Person">

    <!-- Las mascotas de propiedad de esta persona -->
    <Attribute name="pets">
      <type>
        <javaclass name="List">
          <ruleclass name="Pet"/>
        </javaclass>
      </type>
      <derivation>
        <fixedlist>
          <listof>
            <ruleclass name="Pet"/>
          </listof>
          <members>
            <!-- Cada persona tiene exactamente dos mascotas,
                 Skippy y Lassie -->
            <create ruleclass="Pet">
              <!-- establecer el propietario para que sea ESTA persona -->
              <this/>
              <String value="Skippy"/>
              <String value="Kangaroo"/>
            </create>
            <create ruleclass="Pet">
```

```
            <!-- establecer el propietario para que sea ESTA persona -->
            <this/>
            <String value="Lassie"/>
            <String value="Dog"/>
          </create>
        </members>
      </fixedlist>
    </derivation>
  </Attribute>

</Class>

<Class name="Pet">

  <Initialization>
    <Attribute name="owner">
      <type>
        <ruleclass name="Person"/>
      </type>
    </Attribute>
    <Attribute name="name">
      <type>
        <javaclass name="String"/>
      </type>
    </Attribute>
    <Attribute name="species">
      <type>
        <javaclass name="String"/>
      </type>
    </Attribute>
  </Initialization>

</Class>

</RuleSet>
```

Timeline:
Crea una línea de tiempo (consulte “Manejo de datos que cambian a lo largo del tiempo” en la página 51) de un tipo determinado, con valores válidos de fechas especificadas.

Una línea de tiempo debe tener un valor desde el inicio del tiempo (la fecha `null`) y por lo tanto para ayudar con esto, la expresión `Timeline` contiene un elemento `initialvalue` opcional para especificar el valor desde el inicio del tiempo. Si no se utiliza, la colección de “`Intervalo`” en la página 202 utilizada *debe* contener un intervalo con una fecha de inicio `null`, de lo contrario se producirá un error si se evalúa esta expresión en el tiempo de ejecución.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_Timeline"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">

  <Class name="CreateTimelines">

    <!-- Este ejemplo utiliza <initialvalue> para establecer el valor válido
         desde el inicio del tiempo. -->
    <Attribute name="aNumberTimeline">
      <type>
        <javaclass name="curam.creole.value.Timeline">
          <javaclass name="Number"/>
        </javaclass>
      </type>
      <derivation>
```

```
    <Timeline>
      <intervaltype>
        <javaclass name="Number"/>
      </intervaltype>
      <!-- Valor desde el inicio del tiempo -->
      <initialvalue>
        <Number value="0"/>
      </initialvalue>
      <!-- Los intervalos restantes -->
      <intervals>
        <fixedlist>
          <listof>
            <javaclass name="curam.creole.value.Interval">
              <javaclass name="Number"/>
            </javaclass>
          </listof>
          <members>
            <Interval>
              <intervaltype>
                <javaclass name="Number"/>
              </intervaltype>
              <start>
                <Date value="2001-01-01"/>
              </start>
              <value>
                <Number value="10000"/>
              </value>
            </Interval>
            <Interval>
              <intervaltype>
                <javaclass name="Number"/>
              </intervaltype>
              <start>
                <Date value="2004-12-01"/>
              </start>
              <value>
                <Number value="12000"/>
              </value>
            </Interval>

          </members>
        </fixedlist>

      </intervals>
    </Timeline>

  </derivation>
</Attribute>


<!-- Este ejemplo no utiliza <initialvalue>. -->
<Attribute name="aStringTimeline">
  <type>
    <javaclass name="curam.creole.value.Timeline">
      <javaclass name="String"/>
    </javaclass>
  </type>
  <derivation>
    <Timeline>
      <intervaltype>
        <javaclass name="String"/>
      </intervaltype>

      <!-- La lista de intervalos debe incluir uno válido desde la
         fecha nula (inicio de tiempo), de lo contrario, se producirá un error
         en el tiempo de ejecución, si se evalúa esta expresión. -->
      <intervals>
```

```
            <fixedlist>
              <listof>
                <javaclass name="curam.creole.value.Interval">
                  <javaclass name="String"/>
                </javaclass>
              </listof>
              <members>
                <Interval>
                  <intervaltype>
                    <javaclass name="String"/>
                  </intervaltype>
                  <start>
                    <!-- "desde el inicio del tiempo" -->
                    <null/>
                  </start>
                  <value>
                    <String value="Start of time string"/>
                  </value>
                </Interval>
                <Interval>
                  <intervaltype>
                    <javaclass name="String"/>
                  </intervaltype>
                  <start>
                    <Date value="2001-01-01"/>
                  </start>
                  <value>
                    <String value="2001-only String"/>
                  </value>
                </Interval>
                <Interval>
                  <intervaltype>
                    <javaclass name="String"/>
                  </intervaltype>
                  <start>
                    <Date value="2002-01-01"/>
                  </start>
                  <value>
                    <String value="2002-onwards String"/>
                  </value>
                </Interval>
              </members>
            </fixedlist>
          </intervals>
        </Timeline>
      </derivation>
    </Attribute>
  </Class>
</RuleSet>
```

timelineoperation:
Ensambla una línea de tiempo (consulte “Manejo de datos que cambian a lo largo del tiempo” en la página 51) desde llamadas repetidas a una expresión hija. Normalmente `timelineoperation` se utiliza conjuntamente con “`intervalvalue`” en la página 203 y, juntas, estas dos expresiones permiten que otras expresiones operen en valores de líneas de tiempo como si fueran valores primitivos y luego hagan que los datos resultantes se ensamblen en una línea de tiempo.

Consejo: Para cada una de las líneas de tiempo que se utilizan como entrada en el algoritmo, normalmente deberá envolver la expresión que devuelve la línea de tiempo en un “`intervalvalue`” en la página 203 y, a continuación, envolver el resultado global en un `timelineoperation`.

A continuación se proporciona una breve descripción de cómo se comporta timelineoperation en el tiempo de evaluación:

- timelineoperation crea un nuevo contexto de evaluación para hacer el seguimiento de las series de llamadas a realizar a la expresión hija (normalmente la expresión hija se invocará varias veces, para fechas diferentes);
- timelineoperation invoca la única expresión hija con una fecha de contexto de null, que significa el inicio del tiempo;
- durante la evaluación de la expresión hija (y sus dependientes), siempre que se encuentra un "intervalvalue" en la página 203, realiza las acciones siguientes:
  - "intervalvalue" en la página 203 evalúa la única expresión hija para obtener una línea de tiempo y desde esta línea de tiempo obtener el valor en la fecha correspondiente a la fecha actual en el contexto de evaluación de timelineoperation;
  - "intervalvalue" en la página 203 comprueba las demás fechas en las que la línea de tiempo cambia de valor y, para cada una de estas fechas, las añade a una cola de otras fechas en las que timelineoperation debe operar (en una invocación subsiguiente);
- cuando se devuelve el control a timelineoperation, se habrá calculado un valor para una fecha determinada y se habrán identificado las fechas adicionales en las que las líneas de tiempo de entrada cambian de valor. Para cada fecha, timelineoperation vuelve a invocar la expresión hijo (en esta fecha) hasta que no hay más fechas en la cola.

El comportamiento descrito anteriormente significa que las expresiones internas nunca tienen que saber que forman parte del proceso que incluye las líneas de tiempo. Además, el proceso es eficiente porque las expresiones sólo se invocan para fechas en las que las líneas de tiempo de entrada cambian de valor.

Nota: Si timelineoperation opera en una expresión *no* envuelta por un "intervalvalue" en la página 203, la línea de tiempo resultante tendrá un valor constante todo el tiempo.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_timelineoperation"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">

  <Class name="Person">
    <!--
     true durante el tiempo de vida de una persona; false antes de la fecha de nacimiento
     y false de nuevo después de la fecha de nacimiento (si hay alguna)
      -->
    <Attribute name="isAliveTimeline">
      <type>
        <javaclass name="curam.creole.value.Timeline">
          <javaclass name="Boolean"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!--
     Los activos propiedad de la persona, en un momento u otro.  El
     valor de cada activo puede variar con el tiempo.
      -->
    <Attribute name="ownedAssets">
      <type>
```

```
    <javaclass name="List">
      <ruleclass name="Asset"/>
    </javaclass>
  </type>
  <derivation>
    <specified/>
  </derivation>
</Attribute>



<!--
 El valor total de los activos que son propiedad de la persona (o de la
 herencia de la persona, si la persona ha muerto).
  -->
<Attribute name="totalAssetValueTimeline">
  <type>
    <javaclass name="curam.creole.value.Timeline">
      <javaclass name="Number"/>
    </javaclass>
  </type>
  <derivation>
    <!--
     Sumar el valor de todos los activos de propiedad.  El valor de cada
     activo puede cambiar con el tiempo.
      -->

    <!--
     <timelineoperation> creará una línea de tiempo a partir de la serie
     de cálculos de <sum> realizados en ella.

     Cada ejecución de <sum> calculará el total de un
     día determinado; <timelineoperation> ensamblará estos
     totales diarios en una línea de tiempo de números.
      -->
    <timelineoperation>

      <sum>
        <!--
         Para cada activo de propiedad, obtener el tiempo de línea de valor contable.
          -->
        <dynamiclist>
          <list>
            <reference attribute="ownedAssets"/>
          </list>
          <listitemexpression>

            <!--
             Envolver la línea de tiempo devuelta por
             countableValueTimeline, para que <sum> crea
             que está operando en una lista de números, no una
             lista de líneas de tiempo.
              -->
            <intervalvalue>
              <reference attribute="countableValueTimeline">
                <current/>
              </reference>
            </intervalvalue>
          </listitemexpression>
        </dynamiclist>

      </sum>

    </timelineoperation>

  </derivation>
</Attribute>
```

```
<!--
 El punto de límite de corte para estar autorizar a la prestación.  Las personas con
 activos por encima de este umbral variable no están autorizadas para la prestación.

  -->
<Attribute name="maximumAssetsThreshold">
  <type>
    <javaclass name="curam.creole.value.Timeline">
      <javaclass name="Number"/>
    </javaclass>
  </type>
  <derivation>
    <!--
     En una implementación real, este valor tenderá variar
     con el paso del tiempo (por ejemplo de una tabla de tasas).

     Sin embargo, para simplificar el ejemplo esta implementación utiliza
     un <timelineoperation> SIN un <intervalvalue> anidado para
     crear una línea de tiempo que tenga un valor constante todo el tiempo.

     A menudo este uso de <timelineoperation> puede ser útil para
     que las implementaciones ficticias se activen antes en el desarrollo de conjunto de reglas.
    -->

    <timelineoperation>
      <!--
       Valor de constante para siempre de codificación fija - que se deberá sustituir por un
       valor variable más adelante en el desarrollo de reglas.
        -->
      <Number value="10000"/>
    </timelineoperation>
  </derivation>
</Attribute>

<!--
 La persona cualificada para prestación si (en cualquier día determinado)
 la persona está viva y el valor total de los activos de la persona
 no supera el umbral máximo de activo.
  -->
<Attribute name="qualifiesForBenefitTimeline">
  <type>
    <javaclass name="curam.creole.value.Timeline">
      <javaclass name="Boolean"/>
    </javaclass>
  </type>
  <derivation>
    <timelineoperation>
      <all>
        <fixedlist>
          <listof>
            <javaclass name="Boolean"/>
          </listof>
          <members>
            <!--
             operar en las líneas de tiempo como si fueran
             valores primitivos
              -->
            <intervalvalue>
              <reference attribute="isAliveTimeline"/>
            </intervalvalue>

            <compare comparison="&lt;=">
              <intervalvalue>
                <reference attribute="totalAssetValueTimeline"/>
              </intervalvalue>
```

```
                <intervalvalue>
                  <reference attribute="maximumAssetsThreshold"/>
                </intervalvalue>
              </compare>
            </members>
          </fixedlist>

        </all>
      </timelineoperation>

    </derivation>
  </Attribute>



</Class>

<!--
 Un activo, propiedad de una persona en un momento u otro.

 Cada activo se compra y, posteriormente, se puede vender.

 El valor de un activo varía con el tiempo; el activo todavía tiene un
 valor incluso antes o después de ser propiedad de una Persona; sin embargo,
 el valor  _countable_ es cero fuera del periodo de
 propiedad.
  -->
<Class name="Asset">
  <Attribute name="boughtDate">
    <type>
      <javaclass name="curam.util.type.Date"/>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

  <!-- será nulo si el activo no se ha vendido -->
  <Attribute name="soldDate">
    <type>
      <javaclass name="curam.util.type.Date"/>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

  <Attribute name="isOwnedTimeline">
    <type>
      <javaclass name="curam.creole.value.Timeline">
        <javaclass name="Boolean"/>
      </javaclass>
    </type>
    <derivation>
      <existencetimeline>
        <intervaltype>
          <javaclass name="Boolean"/>
        </intervaltype>
        <intervalfromdate>
          <reference attribute="boughtDate"/>
        </intervalfromdate>
        <intervaltodate>
          <reference attribute="soldDate"/>
        </intervaltodate>
        <preExistenceValue>
          <false/>
        </preExistenceValue>
```

```
      <existenceValue>
        <true/>
      </existenceValue>
      <postExistenceValue>
        <false/>
      </postExistenceValue>
    </existencetimeline>

  </derivation>
</Attribute>

<!--
 el valor variable del activo, independientemente de si es
 propiedad de la persona en ese momento
  -->
<Attribute name="valueTimeline">
  <type>
    <javaclass name="curam.creole.value.Timeline">
      <javaclass name="Number"/>
    </javaclass>
  </type>
  <derivation>
    <specified/>
  </derivation>
</Attribute>


<!--
 El valor que cuenta para los activos de la persona - es decir, el
 valor del activo durante el periodo en el que es propiedad,
 de lo contrario 0 cuando no es propiedad.
  -->
<Attribute name="countableValueTimeline">
  <type>
    <javaclass name="curam.creole.value.Timeline">
      <javaclass name="Number"/>
    </javaclass>
  </type>
  <derivation>
    <!--
     volver a ensamblar las salidas de cada invocación de <choose> en una
     línea de tiempo
      -->
    <timelineoperation>
      <choose>
        <type>
          <javaclass name="Number"/>
        </type>
        <when>
          <condition>
            <!--
             operar en cada uno de los intervalos de la
             propiedad constante
            -->

            <intervalvalue>
              <reference attribute="isOwnedTimeline"/>
            </intervalvalue>
          </condition>
          <value>
            <!--
             si en una fecha determinada, el activo es propiedad, entonces
             su valor contable en esa fecha es simplemente su
             valor
              -->
            <intervalvalue>
              <reference attribute="valueTimeline"/>
```

```
              </intervalvalue>
            </value>
          </when>
          <otherwise>
            <value>
              <!--
               si en una fecha determinada, el activo es propiedad, entonces
               su valor contable en esa fecha es cero
                -->
              <Number value="0"/>
            </value>
          </otherwise>
        </choose>

      </timelineoperation>
    </derivation>
  </Attribute>

</Class>

</RuleSet>
```

Consejo: Si una expresión interna devuelve una línea de tiempo y se olvida de envolver esa expresión en un "intervalvalue" en la página 203, verá errores de validación CER como en este ejemplo:

```
<!--
     El valor que cuenta para los activos de la persona - es decir, el
     valor del activo durante el periodo en el que es propiedad,
     de lo contrario 0 cuando no es propiedad.
      -->
    <Attribute name="countableValueTimeline">
      <type>
        <javaclass name="curam.creole.value.Timeline">
          <javaclass name="Number"/>
        </javaclass>
      </type>
      <derivation>
        <!--
         volver a ensamblar las salidas de cada invocación de <choose> en una
         línea de tiempo
          -->
        <timelineoperation>
          <choose>
            <type>
              <javaclass name="Number"/>
            </type>
            <when>
              <condition>
                <!--
                 operar en cada uno de los intervalos de la
                 propiedad constante
                -->

                <!--
                  **** Se ha olvidado de envolver la línea de tiempo devuelta
                  en <intervalvalue> ****
                  -->
                  <reference attribute="isOwnedTimeline"/>
              </condition>
              <value>
                <!--
                 si en una fecha determinada, el activo es propiedad, entonces
                 su valor contable en esa fecha es simplemente su
                 valor
                  -->
                <intervalvalue>
```

```
              <reference attribute="valueTimeline"/>
            </intervalvalue>
          </value>
        </when>
        <otherwise>
          <value>
            <!--
             si en una fecha determinada, el activo es propiedad, entonces
             su valor contable en esa fecha es cero
              -->
            <Number value="0"/>
          </value>
        </otherwise>
      </choose>

    </timelineoperation>
  </derivation>
</Attribute>
```

Error de ejemplo.

```
ERROR ... Example_timelineoperation.xml(276, 19)
  AbstractRuleItem:INVALID_CHILD_RETURN_TYPE: Child 'condition' returns
  'curam.creole.value.Timeline<? extends java.lang.Boolean>',
  but this item requires a 'java.lang.Boolean'.
```

true:
El valor constante booleano “true”.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_true"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="TrueExampleRuleClass">

    <Attribute name="isCuramExpertRulesFantastic">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <true/>
      </derivation>
    </Attribute>

    <Attribute name="didCookbookWinPulitzerPrize">
      <type>
        <javaclass name="Boolean"/>
      </type>
      <derivation>
        <not>
          <true/>
        </not>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

XmlMessage:
Crea un mensaje traducible (consulte “Soporte de localización” en la página 8) a partir de contenido XML de formato libre.

El mensaje creado será el contenido XML literal en el elemento `XmlMessage`, con una excepción: El contenido de cualquier elemento `replace` se establecerá en la expresión que contiene.

El elemento `replace` como mecanismo de sustitución de señal simple; si necesita formato de señal más complejo o la posibilidad de cambiar texto de mensaje sin cambiar reglas, considere la posibilidad de utilizar "`ResourceMessage`" en la página 226 en su lugar.

Nota: Antes de Cúram V6, `XmlMessage` recortaba el espacio en blanco que rodea a los caracteres XML incorporados. A partir de Cúram V6, `XmlMessage` ya no recorta el espacio en blanco y conserva el formato de origen de los caracteres XML (eliminando los comentarios XML).

Si necesita el comportamiento de recorte anterior a Cúram V6 de `XmlMessage`, debe establecer la variable de entorno de aplicación *curam.creole.XmlFormat.enableWhitespaceTrimming* en el valor *true* en el entorno de desarrollo.

En un entorno de producción, debe asegurarse de que si el valor de la variable de entorno *curam.creole.XmlFormat.enableWhitespaceTrimming* cambia dinámicamente, toma las medidas para asegurarse de que se fuerza que los datos obtenidos del sistema que dependen de los atributos de regla que utilizan la expresión `XmlMessage` vuelvan a calcular todas las instancias de valor de atributo almacenadas.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_XmlMessage"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">
  <Class name="XmlMessageExampleRuleClass">

    <Attribute name="emptyMessage">
      <type>
        <javaclass name="curam.creole.value.Message"/>
      </type>
      <derivation>
        <!-- no contiene XML en absoluto -->
        <XmlMessage/>
      </derivation>
    </Attribute>

    <Attribute name="simpleHtmlMessage">
      <type>
        <javaclass name="curam.creole.value.Message"/>
      </type>
      <derivation>
        <!-- La utilización de XmlMessage puede asegurar que
             los elementos XML se inicien y
             finalicen correctamente, por ejemplo <b> y </b> -->
        <XmlMessage>El texto siguiente aparecerá en negrita en un
          navegador: <b>Texto en negrita.</b>
        </XmlMessage>
      </derivation>
    </Attribute>

    <Attribute name="tokenReplacementHtmlMessage">
      <type>
        <javaclass name="curam.creole.value.Message"/>
      </type>
      <derivation>
        <XmlMessage><p/>Este número calculado aparecerá en
```

```
          cursiva y formateado de acuerdo con las preferencias de entorno local:<i>
            <replace>
              <arithmetic operation="+">
                <Number value="1.23"/>
                <Number value="3.45"/>
              </arithmetic>
            </replace>
          </i>
          <p/>Y aquí se muestra un mensaje de recurso: <replace>
            <ResourceMessage key="simpleGreeting"
              resourceBundle="curam.creole.example.Messages"/>
          </replace>
        </XmlMessage>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

Anotaciones

Desde Cúram V6, CER soporta "anotaciones". Una anotación son metadatos adicionales dentro de un conjunto de reglas que están disponibles para los clientes de CER, pero *no* afectan el comportamiento de los cálculos CER.

Los clientes de CER pueden hacer uso de anotaciones para controlar su comportamiento cuando interactúan con conjuntos de reglas CER.

Se pueden colocar anotaciones en estos elementos en un conjunto de reglas CER (aunque cada anotación puede contener la validación para limitar dónde se coloca):

- un conjunto de reglas (consulte “Conjunto de reglas” en la página 158);
- una clase de regla (consulte “Clase de regla” en la página 160);
- un atributo de regla (consulte “Atributo” en la página 161) o
- una expresión (consulte “Expresiones” en la página 162).

Cada elemento de regla puede contener cero, uno o muchas anotaciones. Cada tipo de anotación puede aparecer como máximo una vez en cualquier elemento de regla; por ejemplo, un conjunto de reglas puede contener una anotación `Label` y una anotación `EditorMetadata`, pero no puede contener más de una anotación `Label`.

Listado alfabético de anotaciones completo

Esta sección define todas las anotaciones que se incluyen con la aplicación.

Nota: Algunas anotaciones son para las derivaciones específicas de empresa en la aplicación. Las anotaciones de este tipo todavía se incluyen aquí, pero se remite al lector a otras guías de Cúram que describen esas anotaciones en su contexto empresarial.

`ActiveInEditSuccessionSetPopulation`:
Consulte la `Guía de configuración de asesor de Cúram`.

`Display`:
Consulte la guía `Inside Cúram Eligibility and Entitlement Using Cúram Express Rules`.

`DisplaySubscreen`:

Consulte la guía Inside Cúram Eligibility and Entitlement Using Cúram Express Rules.

EditorMetadata:
Almacena información de diagrama para un conjunto de reglas. Lo mantiene automáticamente el Editor CER.

Esta anotación sólo se puede especificar en un conjunto de reglas (consulte "Conjunto de reglas" en la página 158).

Indexed:
Ya no se utiliza. Sólo se incluye por compatibilidad con versiones anteriores.

Label

Proporciona una descripción localizada de un elemento de conjunto de reglas:

- un conjunto de reglas (consulte "Conjunto de reglas" en la página 158);
- una clase de regla (consulte "Clase de regla" en la página 160);
- un atributo de regla (consulte "Atributo" en la página 161) o
- una expresión (consulte "Expresiones" en la página 162).

La etiqueta contiene un identificador (establecida por el Editor CER) y una descripción (entrada por el usuario).

Cuando un conjunto de reglas CER se guarda o se publica, los valores de las anotaciones de etiqueta en una regla se utilizan para escribir un recurso de propiedad (en el entorno local del usuario) en el almacén de recursos de la aplicación. Y a la inversa, cuando un conjunto de reglas se visualiza en el Editor CER, el recurso de propiedad para la configuración regional del usuario se recupera del almacén de recursos y se utiliza para llenar el valor de las anotaciones de etiqueta en el XML de conjunto de reglas.

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_Label"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">

  <Annotations>
    <!-- Descripción de nivel de conjunto de reglas -->
    <Label name="Ejemplo de conjunto de reglas para etiquetas"
 label-id="annotation1"/>
  </Annotations>
  <Class name="Person">
    <Annotations>
      <!-- Descripción de nivel de clase de regla -->
      <Label name="A Person" label-id="annotation2"/>
    </Annotations>
    <Attribute name="age">
      <Annotations>
        <!-- Descripción de nivel de clase de atributo -->
        <Label name="La edad actual de la persona, en años"
 label-id="annotation3"/>
      </Annotations>
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified>
          <Annotations>
            <!-- Descripción de nivel de expresión -->
```

```
          <Label name="Este valor viene directamente de las pruebas"
 label-id="annotation4"/>
         </Annotations>
       </specified>
     </derivation>
   </Attribute>

   <Attribute name="ageNextBirthday">
     <Annotations>
       <!-- Descripción de nivel de clase de atributo -->
       <Label name="La edad de la persona en el siguiente cumpleaños de la
 persona, en años"
         label-id="annotation5"/>
     </Annotations>
     <type>
       <javaclass name="Number"/>
     </type>
     <derivation>
       <arithmetic operation="+">
         <Annotations>
           <!-- Descripción de nivel de expresión -->
           <Label name="Calcular la edad de la persona en el siguiente cumpleaños"
 label-id="annotation6"/>
         </Annotations>
         <reference attribute="age">
           <Annotations>
             <!-- Descripción de nivel de expresión -->
             <Label name="Obtener la edad actual de la persona"
 label-id="annotation7"/>
           </Annotations>
         </reference>
         <Number value="1">
           <Annotations>
             <!-- Descripción de nivel de expresión -->
             <Label name="El número a añadir para obtener la edad en el siguiente
 cumpleaños" label-id="annotation8"/>
           </Annotations>
         </Number>
       </arithmetic>
     </derivation>
   </Attribute>


  </Class>

</RuleSet>
```

Legislación

Consulte la guía Inside Cúram Eligibility and Entitlement Using Cúram Express Rules.

SuccessionSetPopulation

Consulte la guía Inside Cúram Eligibility and Entitlement Using Cúram Express Rules.

primary

Identifica el atributo primario en una clase de regla, como se designa en el Editor CER.

Esta anotación sólo se puede especificar en una clase de regla (consulte “Clase de regla” en la página 160). El atributo denominado debe ser el nombre exacto de un

atributo declarado en la clase de regla (no se puede utilizar para nombrar un atributo que se hereda, pero no altera temporalmente).

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_primary"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">


  <Class name="Person">
    <Annotations>
      <!-- Declaración del atributo de regla "primary" para esta clase de
 regla, como se muestra en el Editor CER -->
      <primary attribute="age"/>
    </Annotations>
    <Attribute name="age">
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

relatedActiveInEditSuccessionSet

Consulte la Guía de configuración de asesor de Cúram.

relatedEvidence

Consulte la guía Inside Cúram Eligibility and Entitlement Using Cúram Express Rules.

relatedSuccessionSet

Consulte la guía Inside Cúram Eligibility and Entitlement Using Cúram Express Rules.

tags

Asocia códigos de serie arbitrarios con:

- un conjunto de reglas (consulte “Conjunto de reglas” en la página 158);
- una clase de regla (consulte “Clase de regla” en la página 160);
- un atributo de regla (consulte “Atributo” en la página 161) o
- una expresión (consulte “Expresiones” en la página 162).

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="Example_tags"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">

  <Annotations>
    <!-- Códigos de nivel de conjunto de reglas -->
    <tags>
      <tag value="A rule-set tag"/>
      <tag value="Another tag"/>
```

```
    </tags>
  </Annotations>
  <Class name="Person">
    <Annotations>
      <!-- Códigos de nivel de clase de regla-->
      <tags>
        <tag value="A rule-class tag"/>
        <tag value="Another tag"/>
      </tags>
    </Annotations>
    <Attribute name="age">
      <Annotations>
        <!-- Códigos de nivel de clase de atributo -->
        <tags>
          <tag value="A rule-attribute tag"/>
          <tag value="Another tag"/>
        </tags>
      </Annotations>
      <type>
        <javaclass name="Number"/>
      </type>
      <derivation>
        <specified>
          <Annotations>
            <!-- Códigos de nivel de expresión -->
            <tags>
              <tag value="An expression tag"/>
              <tag value="Another tag"/>
            </tags>
          </Annotations>
        </specified>
      </derivation>
    </Attribute>

  </Class>

</RuleSet>
```

Operaciones de lista útiles

La expresión `property` permite la invocación de métodos Java "seguros".

Consulte “Property” en la página 128 y “`property`” en la página 213.

Los conjuntos de reglas contienen normalmente muchas instancias de `java.util.List`.

La lista segura de métodos para `java.util.List` se incluye con CER:

Lista segura para los métodos `java.util.List`.

```
# Lista segura para java.util.List

            contains.safe=true
            containsAll.safe=true


            get.safe=true

            indexOf.safe=true
            isEmpty.safe=true
            lastIndexOf.safe=true
            size.safe=true
            subList.safe=true
```

```
# no expuesto
hashCode.safe=false
listIterator.safe=false
iterator.safe=false
toArray.safe=false

# mutadores - no seguro
add.safe=false
addAll.safe=false
clear.safe=false
remove.safe=false
removeAll.safe=false
retainAll.safe=false
```

Mientras que la descripción de estos métodos está disponible a través del JavaDoc para `java.util.List`, aquí se incluyen las propiedades más útiles habitualmente para referencia del usuario:

- **isEmpty()**

  Devuelve *true* si esta lista no contiene elementos.

- **size()**

  Devuelve el número de elementos en esta lista.

- **get(int index)**

  Devuelve el elemento en la posición especificada de esta lista. Tenga en cuenta que debido a que CER pasa valores numéricos redondeados como instancias de `Número`, deberá utilizar `intValue` para convertir un `Número` en un entero.

- **contains(Object o)**

  Devuelve *true* si esta lista contiene el elemento especificado.

```
<?xml version="1.0" encoding="UTF-8"?>
            <RuleSet name="Example_UsefulListOperations"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:noNamespaceSchemaLocation=
            "http://www.curamsoftware.com/CreoleRulesSchema.xsd">

            <Class name="Person">

            <!-- Se designará exactamente a una persona (en cada unidad familiar)
            como cabeza de familia -->
            <Attribute name="isHeadOfHousehold">
            <type>
            <javaclass name="Boolean"/>
            </type>
            <derivation>
            <specified/>
            </derivation>
            </Attribute>

            <!-- Los hijos de esta persona. -->
            <Attribute name="children">
            <type>
            <javaclass name="List">
            <ruleclass name="Person"/>
            </javaclass>
            </type>
            <derivation>
            <specified/>
            </derivation>
            </Attribute>

            <!-- Indica si esta persona tiene hijos.

            Prueba la propiedad isEmpty de lista. -->
```

```
<Attribute name="hasChildren">
<type>
<javaclass name="Boolean"/>
</type>
<derivation>
<not>
<property name="isEmpty">
<object>
<reference attribute="children"/>
</object>
</property>
</not>
</derivation>
</Attribute>

<Attribute name="numberOfChildren">
<type>
<javaclass name="Number"/>
</type>
<derivation>
<property name="size">
<object>
<reference attribute="children"/>
</object>
</property>
</derivation>
</Attribute>

<!-- El segundo hijo de esta persona, si hay alguno, de lo contrario, nulo -->
<Attribute name="secondChild">
<type>
<ruleclass name="Person"/>
</type>
<derivation>
<!-- Tenemos que comprobar si la persona tiene dos
o más hijos -->
<choose>
<type>
<ruleclass name="Person"/>
</type>
<when>
<condition>
<compare comparison=">=">
<reference attribute="numberOfChildren"/>
<Number value="2"/>
</compare>
</condition>
<value>
<!-- Utilice la propiedad "get" para obtener el segundo elemento
de la lista - indicado por el índice 1 (las listas en
Java se basan en cero) -->
<property name="get">
<object>
<reference attribute="children"/>
</object>
<arguments>
<!-- El número se debe convertir a un entero
(como lo requiere List.subList). -->
<property name="intValue">
<object>
<Number value="1"/>
</object>
</property>

</arguments>
</property>
</value>
```

```
</when>
<otherwise>
<!-- Esta persona no tiene un segundo hijo -->
<value>
<null/>
</value>
</otherwise>
</choose>
</derivation>
</Attribute>

<Attribute name="isChildOfHeadOfHousehold">
<type>
<javaclass name="Boolean"/>
</type>
<derivation>
<property name="contains">
<object>
<!-- Los hijos del cabeza de familia -->
<reference attribute="children">
<!-- recupera el único objeto de regla de persona que
tiene isHeadOfHousehold igual a true-->
<singleitem onEmpty="error" onMultiple="error">
<readall ruleclass="Person">
<match retrievedattribute="isHeadOfHousehold">
<true/>
</match>
</readall>
</singleitem>
</reference>
</object>
<!-- comprobar si la lista de hijos del cabeza de familia
contiene ESTA (THIS) persona -->
<arguments>
<this/>
</arguments>
</property>
</derivation>
</Attribute>

</Class>

</RuleSet>
```

Utilización de CER con el almacén de datos

La aplicación incluye código de integración que puede crear objetos de regla CER a partir de entradas en el almacén de datos de la aplicación. El módulo Universal Access utiliza `DataStoreRuleObjectCreator` para convertir las pruebas recopiladas por un script IEG en objetos de regla CER. Esta información describe como funciona `DataStoreRuleObjectCreator`.

DataStoreRuleObjectCreator

`DataStoreRuleObjectCreator` toma un registro de almacén de datos (normalmente un registro relacionado con un usuario o una persona) y navega a todos los registros descendentes de este registro "raíz" (que normalmente contiene todas las pruebas recopiladas de la persona).

A continuación continúa con la creación de objetos de regla realizando una "correlación natural" directa entre:

- los tipos entidad y los atributos en el esquema de almacén de datos y
- las clases de reglas y el atributo de regla en el conjunto de reglas CER.

DataStoreRuleObjectCreator también actúa de manera especial con los atributos de regla CER con determinados nombres:

- **parentEntity**

  Si una clase de regla contiene un atributo de regla denominado parentEntity, DataStoreRuleObjectCreator establecerá su valor para que sea el objeto de regla creado desde el registro padre en el almacén de datos (si existe). CER emitirá un error de tiempo de ejecución si el tipo de este atributo de regla no coincide con la clase de regla del objeto de regla de la entidad padre; y
- **childEntities_<nombre de clase de regla>**

  Si una clase de regla contiene atributos denominados childEntities_ seguidos del nombre de una clase de regla, DataStoreRuleObjectCreator establecerá el valor de cada atributo de este tipo para que sea una lista de objetos de regla creados desde los registros hijo de ese tipo en el almacén de datos (si existe). CER emitirá un error de tiempo de ejecución si el tipo de este atributo de regla no es una lista de la clase de regla mencionada.

Ejemplo

El comportamiento de DataStoreRuleObjectCreator se explica mejor mediante un ejemplo. Este ejemplo se basa en el módulo de Acceso universal.

Un script IEG puede capturar datos de ingresos para una unidad familiar. La unidad familiar puede contener cualquier número de personas y cada persona puede tener cualquier número de detalles de ingresos.

A continuación se muestra una vista simplificada de la estructura del esquema de almacén de datos:

- Solicitud
  - Persona (0..n)
    - Nombre de pila (serie)
    - Apellidos (serie)
    - Ingresos (0..n)
      - Tipo (código de la tabla de códigos de IncomeType)
      - Importe (Número)

Un ciudadano (Juan) realiza una autoevaluación y registra pruebas para su unidad familiar (sólo Juan y su esposa María) y detalles de ingresos (Juan está desempleado, María tiene dos trabajos a tiempo parcial). Las pruebas de Juan se almacenan como registros en el Almacén de datos:

- Solicitud #1234
  - Persona #1235
    - Nombre: Juan
    - Apellidos: Pérez
    - Ingresos <sin registros>
  - Persona #1236
    - Nombre: María
    - Apellidos: Pérez
    - Ingresos #1238
      - Tipo: Part-time
      - Importe 30

El conjunto de reglas CER configurado para utilizarse con este tipo de evaluación contiene algunas clases de regla, como se indica a continuación (NB no se muestran clases de regla de programa):

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleSet name="DataStoreMappingExample"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
"http://www.curamsoftware.com/CreoleRulesSchema.xsd">

  <!-- NB ninguna clase de regla para el tipo de entidad CDS "Application";
       los atributos almacenados directamente en una aplicación no son
       necesarios para las reglas, por lo que no es necesario crear un objeto de regla
       que no se utilizará. -->

  <!-- El nombre de esta clase de regla coincide con el de un
       tipo de entidad de CDS -->
  <Class name="Person">
    <!-- El nombre de este atributo de regla coincide con el de un
         atributo en el tipo de entidad CDS y, por lo tanto, su valor se
         especificará automáticamente mediante
         DataStoreRuleObjectRetriever. -->
    <Attribute name="firstName">
      <!-- El tipo del atributo de regla debe coincidir con el
           tipo de atributo CDS, de lo contrario CER
           emitirá un error de tiempo de ejecución. -->
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>

    <!-- NB ningún atributo de regla para el atributo CDS para lastName.
-->

    <!-- Los atributos de regla que coincidan con el patrón "childEntities_<nombre de clase de
regla>"
         recibirán un trato especial de
         DataStoreRuleObjectRetriever.

         DataStoreRuleObjectRetriever especificará que el valor de
         este atributo sea todos los objetos de regla creados desde los
         registros de Ingresos hijo que pertenecen al registro de esta persona
         en el CDS. -->
    <Attribute name="childEntities_Income">
      <!-- El tipo debe ser una lista de objetos de regla de ingresos -->
      <type>
        <javaclass name="List">
          <ruleclass name="Income"/>
        </javaclass>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>
  </Class>

  <!-- El nombre de esta clase de regla no coincide con ningún
tipo de entidad CDS,
       por lo tanto DataStoreRuleObjectRetriever no creará ningún objeto de regla
       para esta clase de regla. -->
  <Class name="Benefit">
    <Attribute name="amount">
      <type>
        <javaclass name="Number"/>
```

```
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>
</Class>

<Class name="Income">
  <!-- Un atributo de regla denominado "parentEntity" recibirá
       un trato especial de
       DataStoreRuleObjectRetriever.

       DataStoreRuleObjectRetriever especificará que el valor
       de este atributo sea el objeto de regla creado desde el
       registro de persona que es el padre de este registro de ingresos
       en el CDS. -->
  <Attribute name="parentEntity">
    <!-- El tipo debe ser un solo objeto de regla de persona -->
    <type>
      <ruleclass name="Person"/>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

  <Attribute name="type">
    <type>
      <!-- El tipo de este atributo debe especificar la tabla
           de códigos correcta, coincidiendo con la definición de dominio de CDS. -->
      <codetableentry table="IncomeType"/>
    </type>
    <derivation>
      <specified/>
    </derivation>
  </Attribute>

  <Attribute name="amount">
    <type>
      <javaclass name="Number"/>
    </type>
    <derivation>
      <!-- Esta derivación no se ejecutará nunca, porque
           DataStoreRuleObjectRetriever "especificará" automáticamente
           el valor del correspondiente al del registro CDS;
           una vez que se ha especificado un valor CER no intenta
           calcularlo.

           En general, los atributos que esperan llenarse
           mediante DataStoreRuleObjectRetriever se deben marcar
           como <specified/> para evitar cualquier confusión entre
           las pruebas autónomas de los conjuntos de reglas y las pruebas
           con DataStoreRuleObjectRetriever.
      -->
      <Number value="123"/>
    </derivation>
  </Attribute>

  <!-- Este atributo no está presente en el tipo de entidad de CDS,
       por lo que no se llenará.  Esto es exactamente lo que
       deseamos, porque su valor se obtiene de otros
       atributos de regla de la forma CER normal -->
  <Attribute name="isCountable">
    <type>
      <javaclass name="Boolean"/>
    </type>
    <derivation>
```

```
        <choose>
          <type>
            <javaclass name="Boolean"/>
          </type>
          <test>
            <reference attribute="type"/>
          </test>
          <when>
            <condition>
              <Code table="IncomeType">
                <String value="Full-time"/>
              </Code>
            </condition>
            <value>
              <true/>
            </value>
          </when>
          <when>
            <condition>
              <Code table="IncomeType">
                <String value="Part-time"/>
              </Code>
            </condition>
            <value>
              <true/>
            </value>
          </when>
          <otherwise>
            <value>
              <false/>
            </value>
          </otherwise>
        </choose>
      </derivation>
    </Attribute>

    <!-- Este atributo de regla no se calcula ni
         corresponde a un atributo en el
         tipo de entidad de CDS.

         Si se hace referencia al valor de este atributo
         en el tiempo de ejecución, CER informará de un error de tiempo de ejecución:
         "El valor debe especificarse antes de utilizarse
          (no se puede calcular)."
    -->
    <Attribute name="employerName">
      <type>
        <javaclass name="String"/>
      </type>
      <derivation>
        <specified/>
      </derivation>
    </Attribute>
  </Class>

</RuleSet>
```

Cuando Juan completa su autoevaluación, el módulo de acceso universal carga el conjunto de reglas CER anterior y crea una sesión CER.

El Módulo de acceso universal llama a `DataStoreRuleObjectCreator` y especifica el registro del almacén de datos raíz (Application #1234).

`DataStoreRuleObjectCreator` recupera todos los registros de descendente de Application #1234 y los procesa de la manera siguiente:

- **Application #1234**

  Se omite, porque no hay ninguna clase de regla denominada "Application" en el conjunto de reglas;
- **Person #1235**

  crea una instancia de objeto de regla de la clase de regla `Person`, con:
  - **firstName**

    Se especifica que sea "Juan";
  - **lastName**

    Se omite, porque no hay ningún atributo de regla denominado "lastName" en la clase de regla `Person`;
  - **childEntities_Income**

    Se especifica que sea una lista vacía, porque no hay ningún registro de ingresos para el registro de Person #1235;
- **Person #1236**

  crea una instancia de objeto de regla de la clase de regla `Person`, con:
  - **firstName**

    Se especifica que sea "María";
  - **lastName**

    Se omite;
  - **childEntities_Income**

    Se especifica que sea una lista que contenga dos objetos de regla de ingresos de María (creados más abajo);
- **Income #1237**

  crea una instancia de objeto de regla de la clase de regla `Income`, con:
  - **parentEntity**

    Se especifica que sea el objeto de regla `Person` creado (más arriba) para María desde el registro de Person #1236;
  - **type**

    Se especifica que sea el código "Part-time";
  - **amount**

    Se especifica que sea el número "25";
  - **(employerName)**

    No se especifica, porque no hay ningún atributo denominado "employerName" en la entidad de almacén de datos;
- **Income #1238**

  crea una instancia de objeto de regla de la clase de regla `Income`, con:
  - **parentEntity**

    Se especifica que sea el objeto de regla `Person` creado (más arriba) para María desde el registro de Person #1236;
  - **type**

    Se especifica que sea el código "Part-time" y
  - **amount**

    Se especifica que sea el número "30".

Por último, el Módulo de acceso universal hace las preguntas al conjunto de reglas las preguntas estándares sobre los programas, la elegibilidad y explicación; las

clases de reglas para los programas (no se muestran en el ejemplo anterior) acceden a los objetos de regla (creados por `DataStoreRuleObjectCreator`) al responder a estas preguntas.

Para obtener más información sobre el almacén de datos de la aplicación y sus esquemas, consulte la guía `Creación de esquemas de almacén de datos`.

Conformidad para CER

Una descripción sobre cómo desarrollar reglas CER de una forma compatible. Seguir estas consideraciones hace que sea más fácil actualizar a versiones futuras de la aplicación.

API pública de CER

La infraestructura de CER tiene una API pública que puede invocar en las pruebas de conjunto de reglas y el código de aplicación. Los ejemplos de este manual muestran cómo utilizar muchas características de la API pública de infraestructura de CER. La aplicación no cambiará ni eliminará nada en esta API pública sin seguir los estándares para manejar el impacto del cliente.

A menos que se permita explícitamente en el JavaDoc, *no* debe proporcionar su propia implementación de cualquier interfaz Java de CER ni subclasificar ninguna clase Java de implementación de CER.

Identificación de la API

El JavaDoc que se incluye con CER es el único medio de identificar qué clases de público, interfaces y métodos forman la API pública CER.

Fuera de la API

La infraestructura CER también contiene algunas clases, interfaces y métodos públicos, que *no* forman parte de la API.

Importante: Para satisfacer los requisitos, *no* debe crear ninguna dependencia en ninguna clase o interfaz CER, ni llamar a ningún método que no sean los descritos en el JavaDoc.

Las clases, interfaces y métodos CER fuera de la API pública están sujetos a cambios o se pueden retirar sin previo aviso.

A menos que se permita explícitamente en el JavaDoc, *no* debe poner ninguna de sus propias clases o interfaces en el paquete `curam.creole` o cualquiera de sus subpaquetes.

Expresiones CER

Sólo se soportan las expresiones CER que se listan en este manual de instrucciones. Consulte “Listado alfabético completo de expresiones” en la página 164 para conocer las expresiones soportadas.

Tenga en cuenta que el esquema de conjunto de reglas incluido con CER también contiene algunas expresiones no soportadas. Estas expresiones son experimentales por naturaleza y están sujetas a cambio o eliminación sin previo aviso. No utilice ninguna expresión CER no soportada en los conjuntos de reglas.

Conjuntos de reglas incluidos con la aplicación

Algunos componentes de la aplicación pueden incluir conjuntos de reglas CER que pueden tener sus propios requisitos de conformidad. Estos requisitos de conformidad para los conjuntos de reglas CER incluidos por los componentes de aplicación están fuera del ámbito de este documento.

Consulte la documentación que acompaña a esos componentes para conocer si está autorizado a personalizar el conjunto de reglas CER y, si es así, qué restricciones de personalización se aplican.

Los clientes no deben modificar el RootRuleSet que se incluye con CER.

Ejemplos de esta guía

Los artefactos de ejemplo de esta guía (conjuntos de reglas, archivos de propiedades y código Java) están sujetos a cambio o se pueden retirar sin previo aviso.

Puede *copiar* libremente estos artefactos de ejemplo para su propio uso; sin embargo, tenga en cuenta que no se proporciona soporte de actualización para estos artefactos de ejemplo.

Tablas de base de datos de CER

La infraestructura de CER incluye una serie de tablas de base de datos. En general, estas tablas son internas a CER y los datos sobre ellas sólo se pueden ser leer o escribir a través de la API pública de CER.

Para obtener más detalles sobre las restricciones de lectura/escritura para estas tablas de base de datos, consulte las subsecciones siguientes.

Nota: Todas las tablas de base de datos de infraestructura de CER tienen como prefijo la palabra CREOLE; sin embargo, lo contrario no es cierto.

Hay tablas con el prefijo CREOLE que forman parte de otros componentes de aplicación y que están sujetos a sus propias declaraciones de conformidad. A continuación, sólo se describen las declaraciones de conformidad para tablas de base de datos de *Infraestructura de CER*.

CREOLERuleSet

Esta tabla de base de datos almacena una fila para cada conjunto de reglas CER publicado en el sistema.

Normalmente el acceso de lectura y grabación a esta tabla de base de datos está restringido sólo a la API pública de CER; sin embargo, se le permite utilizar el gestor de datos de la aplicación para llenar esta tabla de base de datos a condición de que se cumplan los criterios siguientes:

- El valor de la columna `name` debe coincidir con el nombre de conjunto de reglas definido en el XML para la columna `ruleSetDefinition` y
- El valor de la columna `ruleSetVersion` debe ser `null`.

Esto es una entrada de ejemplo de un archivo `CREOLERuleSet.dmx` compatible.

```
<?xml version="1.0" encoding="UTF-8"?>
<table name="CREOLERULESET">
<column name="creoleRuleSetID" type="id"/>
<column name="name" type="text"/>
<column name="ruleSetDefinition" type="blob"/>
```

```
<column name="versionNo" type="number" />
<column name="ruleSetVersion" type="number"/>
<row> ... </row>
<row>
<attribute name="creoleRuleSetID">
<!-- Utilizar algunos identificadores exclusivos apropiados -->
<value>99999</value>
</attribute>
<attribute name="name">
<!-- Este nombre debe coincidir con el valor <RuleSet name="..."> 
del XML del conjunto de reglas contenido en ruleSetDefinition
siguiente -->
<value>MyRuleSet</value>
</attribute>
<attribute name="ruleSetDefinition">
<value>./path/to/MyRuleSet.xml</value>
</attribute>
<attribute name="ruleSetVersion">
<!-- Debe ser un elemento <value/> vacío, que significa un valor de
base de datos NULL -->
<value/>
</attribute>
<attribute name="versionNo">
<!-- initial optimistic lock versionNo value -->
<value>1</value>
</attribute>
</row>
<row> ... </row>
</table>
```

Consulte la guía Trabajar con Cúram Express Rules para conocer los pasos para extraer los datos de CREOLERuleSet de la aplicación (y los datos AppResource acompañantes, si es necesario).

CREOLEMigrationControl

CREOLEMigrationControl es una tabla de control de una sola fila que se utiliza para evitar la publicación simultánea de conjuntos de reglas CER.

Los datos de esta tabla son internos a CER y cualquier otro componente no los puede leer o escribir.

La única fila de esta tabla se llena mediante un archivo DMX incluido con la aplicación. Los clientes no deben modificar este archivo DMX ni crear otros archivos DMX destinados a la tabla CREOLEMigrationControl.

Otras tablas de base de datos de infraestructura de CER

Las tablas de base de datos restantes que se incluyen con la infraestructura de CER son:

- CREOLEAttributeAvailability;
- CREOLEAttributeInheritance;
- CREOLERuleAttribute;
- CREOLERuleAttributeValue;
- CREOLERuleClass;
- CREOLERuleClassInheritance;
- CREOLERuleObject;
- CREOLERuleSetDependency;
- CREOLERuleSetEditAction;
- CREOLERuleSetSnapshot; and

- CREOLEValueOverflow.

Estas tablas de base de datos de infraestructura de CER están dentro de esta condición de conformidad general: los datos de estas tablas de base de datos no se deben leer o grabar de otro modo que no sea a trabés de la API pública de CER. En concreto, *no* se soporta el llenado inicial de estas tablas de base de datos a través de archivos DMX.

Gestor de dependencias

El Gestor de dependencias es interno de Cúram y no se soporta ningún acceso desde el código personalizado o cualquier personalización.

Todos los artefactos que pertenecen al gestor de dependencias están incluidos en el paquete de código `curam.dependency` y sus subpaquetes. CER contribuye con algunos artefactos de esta paquete de código y otros son de la aplicación principal. No debe colocar ninguna de sus propias clases o interfaces en el paquete de código `curam.dependency` o ninguno de sus subpaquetes.

El Gestor de dependencias es propietario de las tablas de base de datos siguientes:

- Dependency;
- PrecedentChangeSet;
- PrecedentChangeItem; and
- PrecedentChangeSetBatchCtrl.

Estas tablas de base de datos no se deben personalizar de ninguna forma y los datos de estas tablas de base de datos no las debe leer o grabar nada que no sea el propio Gestor de dependencias.

El llenado inicial de estas tablas de base de datos utilizando archivos DMX *no* está soportado, con la excepción de los archivos DMX incluidos con la aplicación; además, no está soportado personalizar u omitir el llenado de estas tablas de base de datos utilizando los archivos DMX incluidos con la aplicación.

Avisos

Esta información se ha desarrollado para productos y servicios ofrecidos en los Estados Unidos. Es posible que IBM no ofrezca los productos, servicios o características que se describen en este documento en otros países. Póngase en contacto con el representante local de IBM para obtener información acerca de los productos y servicios que actualmente están disponibles en su zona. Las referencias a programas, productos o servicios de IBM no pretenden establecer ni implicar que sólo puedan utilizarse dichos productos, programas o servicios de IBM. En su lugar, se puede utilizar cualquier producto, programa o servicio funcionalmente equivalente que no infrinja los derechos de propiedad intelectual de IBM. Sin embargo, es responsabilidad del usuario evaluar y verificar el funcionamiento de cualquier producto, programa o servicio que no sea de IBM. IBM puede tener patentes o aplicaciones pendientes de patente que conciernan al tema descrito en este documento. La entrega de este documento no le otorga ninguna licencia sobre dichas patentes.. Puede enviar consultas sobre licencias, por escrito, a:

IBM Director of Licensing

IBM Corporation

North Castle Drive

Armonk, NY 10504-1785

U.S.A.

Para consultas sobre licencias relativas a la información de doble byte (DBCS), póngase en contacto con el departamento de propiedad intelectual de IBM en su país o envíe las consultas, por escrito, a:

Intellectual Property Licensing

Legal and Intellectual Property Law.

IBM Japan Ltd.

19-21, Nihonbashi-Hakozakicho, Chuo-ku

Tokyo 103-8510, Japón

El siguiente párrafo no se aplica al Reino Unido ni a ningún otro país en las que tales provisiones sean incompatibles con la legislación local: INTERNATIONAL BUSINESS MACHINES CORPORATION PROPORCIONA ESTA PUBLICACIÓN "TAL CUAL" SIN GARANTÍA DE NINGÚN TIPO, SEA EXPRESA O IMPLÍCITA, INCLUIDAS, AUNQUE SIN LIMITARSE A ELLAS, LAS GARANTÍAS IMPLÍCITAS DE NO CONTRAVENCIÓN, COMERCIABILIDAD O IDONEIDAD PARA UN PROPÓSITO DETERMINADO. Algunos estados no permiten la renuncia de garantías explícitas o implícitas en determinadas transacciones, por lo que es posible que este párrafo no se aplique en su caso.

Esta información puede contener imprecisiones técnicas o errores tipográficos. Periódicamente se efectuarán cambios en la información aquí contenida; dichos

cambios se incorporarán en las nuevas ediciones de la publicación. BM puede realizar mejoras o cambios en los productos o programas descritos en esta publicación en cualquier momento y sin previo aviso.

Las referencias en esta información a sitios web que no son de IBM se proporcionan sólo para su comodidad y de ninguna manera constituyen una aprobación de estos sitios web. Los materiales de estos sitios Web no forman parte de los materiales de IBM para este producto y el uso que se haga de estos sitios Web es de la entera responsabilidad del usuario.

IBM puede utilizar o distribuir la información que se le suministre del modo que estime oportuno, sin incurrir por ello en ninguna obligación con el remitente. Los titulares de licencias de este programa que deseen tener información sobre el mismo con el fin de: (i) intercambiar información entre programas creados de forma independiente y otros programas (incluido éste) y (ii) utilizar mutuamente la información que se ha intercambiado, deberán ponerse en contacto con:

IBM Corporation

Dept F6, Bldg 1

294 Route 100

Somers NY 10589-3216

U.S.A.

Dicha información puede estar disponible, sujeta a los términos y condiciones apropiados, incluyendo en algunos casos el pago de una tasa.

El programa bajo licencia que se describe en este documento y todo el material bajo licencia que se encuentra disponible para el programa se proporcionan de acuerdo con los términos del Acuerdo del Cliente de IBM, el Acuerdo Internacional de Licencia de Programas o cualquier acuerdo equivalente entre IBM y el Cliente.

Cualquier dato relacionado con el rendimiento que aquí se presente se ha obtenido en un entorno controlado. Por lo tanto, los resultados obtenidos en otros entornos operativos pueden variar significativamente. Es posible que algunas medidas se hayan tomado en sistemas que se están desarrollando y no se puede garantizar que dichas medidas serán iguales en los sistemas disponibles en general. Además, es posible que algunas mediciones se haya estimado mediante extrapolación. Los resultados reales pueden variar. Los usuarios de este documento deben verificar los datos aplicables a su entorno específico.

La información referente a productos que no son de IBM se ha obtenido de los proveedores de esos productos, de sus anuncios publicados o de otras fuentes disponibles.

IBM no ha probado tales productos y no puede confirmar la precisión de su rendimiento, su compatibilidad ni ningún otro aspecto relacionado con productos que no son de IBM. Las preguntas relacionadas con las posibilidades de los productos que no son de IBM deben dirigirse a los proveedores de tales productos.

Todas las sentencias relativas a la dirección o intención futura de IBM están sujetas a modificación o retirada sin previo aviso, y sólo representan objetivos.

Todos los precios de IBM que se muestran son precios actuales de venta al por menor sugeridos por IBM y están sujetos a modificaciones sin previo aviso. Los precios del intermediario podrían variar.

Esta información se utiliza a efectos de planificación. iar antes de que los productos descritos estén disponibles.

Esta información contiene ejemplos de datos e informes utilizados en operaciones comerciales diarias. Para ilustrarlas de la forma más completa posible, los ejemplos pueden incluir nombres de personas, empresas, marcas y productos. Todos estos nombres son ficticios y cualquier similitud con nombres y direcciones utilizados por una empresa real es totalmente fortuita.

LICENCIA DE COPYRIGHT:

Esta información contiene programas de aplicación de ejemplo en lenguaje fuente que ilustran técnicas de programación en diversas plataformas operativas. Puede copiar, modificar y distribuir los programas de ejemplo de cualquier forma, sin tener que pagar a IBM, con intención de desarrollar, utilizar, comercializar o distribuir programas de aplicación que estén en conformidad con la interfaz de programación de aplicaciones (API) de la plataforma operativa para la que están escritos los programas de ejemplo. Estos ejemplos no se han probado en profundidad bajo todas las condiciones. En consecuencia, IBM no puede garantizar ni afirmar la fiabilidad, utilidad o funcionalidad de estos programas. Los programas de ejemplo se proporcionan "TAL CUAL", sin ningún tipo de garantía. IBM no asumirá ninguna responsabilidad por daños ocasionados por el uso de los programas de ejemplo.

Cada copia o parte de estos programas de ejemplo o cualquier trabajo derivado de los mismos, debe incluir un aviso de copyright como el siguiente:

© (nombre de la empresa) (año). Algunas partes de este código se derivan de programas de ejemplo de IBM Corp.

© opyright IBM Corp. _especifique el año o años_. Reservados todos los derechos.

Si visualiza esta información en una copia software, es posible que no aparezcan las fotografías ni las ilustraciones en color.

Consideraciones sobre la política de privacidad

Los productos de IBM Software, incluidas las soluciones de software como servicio (“Ofertas de software”) pueden utilizar cookies u otras tecnologías para recabar información de uso del producto, ayudar a mejorar la experiencia del usuario final, adaptar las interacciones con el usuario final u otros fines. En muchos casos, las Ofertas de software no recopilan información de identificación personal. Algunas de nuestras Ofertas de software pueden ayudar a recabar información de identificación personal. Si esta Oferta de software utiliza cookies para recabar información de identificación personal, a continuación se expone información específica sobre el uso de cookies de esta oferta.

Dependiendo de las configuraciones desplegadas, esta Oferta de software podrá utilizar cookies de sesión u otras tecnologías similares que recaben el nombre, la contraseña u otra información de identificación personal a efectos de gestión de la sesión, autenticación, usabilidad de usuario mejorada, configuración de un inicio

de sesión único u otros fines de seguimiento del uso y/o funcionales. Dichas cookies o tecnologías similares no se pueden inhabilitar.

Si las configuraciones desplegadas para esta Oferta de software le proporcionan a usted como cliente la capacidad de recabar información de identificación personal de usuarios finales por medio de cookies y otras tecnologías, deberá buscar su propio asesoramiento legal relativo a las leyes aplicables a dicha recopilación de datos, incluyendo cualquier requisito de aviso y consentimiento.

Para obtener información adicional relativa al uso de diversas tecnologías, incluidas las cookies, a tales fines, consulte la política de privacidad de IBM en http://www.ibm.com/privacy y la declaración de privacidad en línea de IBM en http://www.ibm.com/privacy/details, las secciones tituladas “Cookies, balizas web y otras tecnologías” y “Declaración de privacidad de los productos software y del software como servicio de IBM” en http://www.ibm.com/software/info/product-privacy.

Marcas registradas

IBM, el logotipo de IBM e ibm.com son marcas registradas de International Business Machines Corp., registradas en muchas jurisdicciones en todo el mundo. Otros nombres de productos y servicios pueden ser marcas registradas de IBM u otras empresas. Encontrará una lista actual de marcas registradas de IBM en la web en "Copyright and trademark information" en http://www.ibm.com/legal/us/en/copytrade.shtml.

Otros nombres pueden ser marcas registradas de sus respectivos propietarios. Otros nombres de empresas, productos o servicios pueden ser marcas registradas o de servicio de terceros.

Impreso en España