DaFRule: Un Modelo de Reglas Enriquecido mediante Flujos de Datos para la Definición Visual del Comportamiento Reactivo de Entidades Virtuales

Patricia Pons, Alejandro Catala, Javier Jaen, Jose A. Mocholi

Grupo ISSI, Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València
Camino de Vera s/n, 46022, Valencia
{ppons, acatala, fjaen, jmocholi}@dsic.upv.es

Abstract. Pese a que los entornos y lenguajes específicamente diseñados para programadores noveles deberían ser de utilidad, programar todavía resulta complicado para adolescentes no expertos. Más aún si se trata de juegos reactivos basados en reglas, en los que el usuario marca el hilo de la historia. Para este tipo de entornos en los que escribir código para definir el comportamiento de las entidades virtuales con suficiente expresividad requiere experiencia, sería deseable proveer mecanismos que faciliten dicha tarea. Aprovechando que tecnologías tales como las superficies interactivas han introducido maneras más naturales de interactuar con los sistemas, este artículo propone un nuevo lenguaje visual basado en reglas, que permita definir el comportamiento reactivo de juegos 2D mediante flujos de datos, sin limitar la expresividad de los mismos. Para ello, tras una evaluación con sujetos adolescentes sobre la comprensión del lenguaje de flujos de datos, se ha definido el correspondiente modelo para la implementación de un editor de reglas reactivas, así como la formalización de dicho lenguaje. Todo ello ha sido validado con la construcción de un prototipo del editor y de un depurador, que además han permitido detectar necesidades que deberán ser abordadas en la implementación del editor para una superficie interactiva.

Keywords: Reglas; ECA; Eventos; Flujos de datos; Lenguaje Visual; Tecnología de juego; Entidades virtuales; Superficie Interactiva; Interfaces de Usuario Tangibles; ITUs.

1 Introducción

Las nuevas generaciones están muy familiarizadas con la tecnología y los avances en computación. La mayoría de los jóvenes de hoy en día son capaces de manejar multitud de aplicaciones de ordenador con fluidez, especialmente juegos educativos interactivos o videojuegos, y más aún, comprender el comportamiento de los mismos. Esta comprensión provoca el desarrollo del pensamiento computacional, habilidad que permite resolver problemas lógicos de diferentes disciplinas mediante la abstracción de los mismos en términos computacionales. Precisamente por su

implicación en diferentes áreas de forma generalista, el pensamiento computacional es considerado como una habilidad de suma importancia a ser explotada para el desarrollo humano [1]. Sin embargo, a pesar de estas habilidades, programar como tal todavía resulta complicado para principiantes o no-programadores.

A pesar de que la aparición de los lenguajes visuales ha contribuido significativamente a disminuir las barreras de la programación, como se indica en [2], la interacción con las interfaces de dichos lenguajes visuales sigue estando limitada a aplicaciones de escritorio. Sin embargo, avances recientes en el campo de la investigación de la interacción hombre-máquina (HCI) han traído nuevas plataformas y técnicas de interacción más naturales que aportan algunas ventajas adicionales en la dimensión social y colaborativa. En el caso de las Interfaces Tangibles de Usuario (ITUs), y en particular en las superficies interactivas, el usuario interactúa con el sistema mediante sus manos o dedos, o bien mediante la manipulación de objetos físicos que han sido especialmente diseñados para tal fin. Mientras que la interacción con los dedos es extremadamente intuitiva, puede estar en ocasiones limitada en ciertas acciones. Es por ello que estas interfaces permiten interactuar con objetos físicos para obtener muchas más posibilidades de interacción. Dichos objetos físicos pueden asociarse con otros elementos de la superficie, moverse, girarse, etc., provocando cambios en el sistema y en las entidades del mismo. Además, muchos de estos dispositivos soportan múltiples interacciones simultáneamente. Por tanto, nos encontramos con interfaces que aúnan la colaboración e interacción entre usuarios, el trabajo cooperativo, y la interacción basada en juegos [3][4][5].

Con el fin de contribuir al desarrollo futuro de una plataforma de juegos basada en una superficie interactiva que dé soporte al aprendizaje creativo y el desarrollo del pensamiento computacional, este artículo se centra en dos aspectos principales relacionados con la programación de entidades en ecosistemas virtuales. Por un lado, se propone la especificación del comportamiento reactivo de las entidades en forma de reglas. Este concepto es ampliamente usado en tecnología de juegos y algunos estudios han revelado que también es ampliamente usado y mejor comprendido por gente joven o no-programadores [6][7]. Por otro lado, la expresividad del lenguaje basado en reglas propuesto debe permitir cálculos no triviales. Los usuarios que diseñen un ecosistema reactivo deben ser capaces de crear expresiones complejas para definir valores tanto de las propiedades de las entidades del ecosistema como de los argumentos de las acciones involucradas.

Teniendo en cuenta lo anterior, en este artículo se presenta, en primer lugar, un modelo para un lenguaje visual basado en descripciones de flujos de datos para soportar la definición de reglas reactivas en superficies interactivas por jóvenes no-programadores. En segundo lugar, se presenta un prototipo que permite la edición de reglas basadas en el modelo anterior para validar la viabilidad del mismo y como paso previo a la implementación definitiva de un editor de reglas en una interfaz tangible multitáctil. Además este lenguaje visual podría ser aplicado a muchos de los entornos en los que se requeriría definir comportamiento reactivo de las entidades que intervienen, proveyendo, pese a la utilización de metáforas visuales, una alta expresividad, de la que habitualmente carecen los entornos basados en técnicas visuales en comparación a otros basados en lenguajes textuales. Asimismo dicho prototipo ha permitido la detección de necesidades y dificultades en el diseño que requerirá el editor de reglas colaborativo para una superficie interactiva.

El resto del artículo se organiza como se indica a continuación. La sección 2 describe algunos trabajos relacionados de interés. La sección 3 introduce el modelo de reglas reactivas empleado, mostrando algunos ejemplos de la expresividad permitida por dicho modelo. La sección 4 reporta un estudio preliminar acerca del uso de flujos de datos por no-programadores. La sección 5 describe el prototipo implementado para el editor de reglas propuesto. Finalmente, se concluye resaltando los aspectos más relevantes de nuestro trabajo, los logros obtenidos y el futuro trabajo a desarrollar.

2 Trabajos Relacionados

Actualmente existen varios entornos basados en juegos que emplean cierto tipo de lenguaje de programación para expresar el comportamiento de sus personajes o entidades virtuales, o la evolución de un mundo virtual al completo. Los objetivos, tecnologías o paradigmas de programación empleados en estos entornos pueden variar, así como la expresividad permitida por cada uno, pero todos ellos están enfocados a no-programadores o a adolescentes que inician sus estudios en informática. Uno de estos entornos es Scratch [8], enfocado a la producción multimedia, e inspirado en el lenguaje visual LogoBlocks [9] para la construcción de bloques virtuales. Sin embargo, a pesar de que es uno de los entornos más visuales, su expresividad está más limitada que aquellos entornos que no son tan visuales o que recaen en lenguajes puramente textuales. No ocurre lo mismo con Alice [10], uno de los entornos más destacados. Alice emplea un lenguaje altamente expresivo mediante el paradigma de programación estructurada por selección. Se ha demostrado que con este método los usuarios sin conocimientos de programación son capaces de programar porciones de código bastante extensas. En el ámbito de la simulación, uno de los trabajos relevantes es Agentsheets [11], que emplea un lenguaje basado en reglas para especificar el comportamiento de agentes simulables. Todos estos entornos se basan en aplicaciones de escritorio, con métodos de entrada conocidos y básicos como teclado y ratón, lo cual facilita sin duda su desarrollo.

Existen otros trabajos de interés no basados necesariamente en reglas, pero que tratan de ampliar horizontes, y son por ello de interés. Turtan es un lenguaje de programación tangible que combina con éxito los principales conceptos de Logo con los mecanismos de interacción de las superficies interactivas [12]. AlgoBlocks es también un lenguaje de programación tangible, enfocado a mejorar y facilitar la interacción entre los aprendices [13], algo que no es tan fácil de lograr en sistemas de escritorio. Por último, Tern es otro lenguaje de programación tangible que considera la interacción tangible como una alternativa efectiva a los lenguajes textuales y visuales de propósito docente, en el cual la colaboración y cooperación resultan de sumo interés [14].

3 Comportamiento Reactivo con Reglas en DaFRULE

Las aplicaciones y sistemas orientados a no-programadores cuyo objetivo es emplear la programación para lograr otros medios, están en su mayoría basados en eventos, tal

y como se revisa en [2]. Eventos y reglas son, por tanto, los componentes básicos necesarios que permiten expresar comportamientos en dichos sistemas. Una aproximación basada en eventos hace más comprensible la definición de las reglas que establecen dicho comportamiento, tal y como demuestran los resultados de los estudios experimentales en [6][7]. Es por ello que en el presente trabajo se ha adoptado esta aproximación, ya que parece la más razonable para expresar comportamiento reactivo en un mundo virtual.

Los eventos son la pieza fundamental de esta aproximación. Un ecosistema virtual está poblado de entidades. Éstas son instancias de una definición de entidad, que especifica tanto su apariencia visual como su comportamiento. En otras palabras, las entidades conforman a un tipo de entidad determinado. En cierto modo, estas entidades del ecosistema han sido dotadas de naturaleza orientada a objetos, por lo que tienen propiedades y acciones de acuerdo a la especificación de los mencionados tipos de entidad a los que pertenecen. Además, durante la simulación, son capaces de inyectar ocurrencias de eventos en el sistema a medida que transcurre ésta. Dichas ocurrencias corresponden a una definición de un tipo de evento. Las ocurrencias de evento lanzadas por las entidades durante la simulación harán evolucionar el estado del ecosistema a medida que las reglas especificadas se instancien en consecuencia.

Para aumentar la expresividad del lenguaje, las reglas se han considerado a nivel del ecosistema, no ligadas a ninguna entidad específica. De este modo, se pueden definir reglas en las que intervienen poblaciones de entidades de manera más natural, en lugar de tener que definir una regla para cada una de las entidades de una colección.

En este contexto consideramos reglas definidas formalmente como un par ordenado R = <P, Q>, donde P es el antecedente y Q el consecuente.

El antecedente, P, se define como P = (E, S, C) donde E es la definición de un evento que debe ser lanzado por una fuente S (p.ej. una entidad virtual), y C es la condición que deben cumplir ciertos datos (p.ej. propiedades y atributos) del evento E y la fuente S para que la regla se instancie.

El consecuente, Q, se define como Q = (T, O, F, {DF}), donde T es la población destino a la que afecta la regla, O es la operación a ejecutar sobre cada una de las entidades de la población destino, F es una condición que filtra las entidades de la población destino que se verán afectadas por la operación anterior, y {DF} es un conjunto de flujos de datos que especifican cómo se establecen los parámetros de la operación antes de la ejecución.

La operación O del consecuente de la regla podrá ser, o bien ejecutar una acción, o bien asignar valor a una propiedad de la población destino. Para el caso en el que se pretenda asignar valor a una propiedad, tan sólo es necesario un único flujo de datos DF que especifique cómo calcular el valor a establecer en la propiedad. En cambio, si la operación es la invocación de una acción, habrá tantos flujos de datos en el conjunto {DF} como parámetros tenga la acción, un flujo por cada parámetro. Las condiciones C y F se expresan también mediante flujos de datos, en los cuales el valor resultante debe ser booleano, y se empleará en simulación para determinar si la condición se satisface o no.

La semántica para una regla R es la siguiente: si una entidad perteneciente a S lanza un evento de tipo E, y se cumple la condición C, entonces la regla es instanciada y lanzada. La operación O se ejecutará sobre aquellas entidades de T que cumplan la

condición F, estableciendo los parámetros de O conforme a la especificación de los flujos de datos {DF}. La estructura de la regla formalizada se muestra en la Figura 1.

SI/CUANDO S LANZA una ocurrencia de evento de tipo E Y SE SATISFACE la condición C
ENTONCES
    PARA CADA entidad en T QUE CUMPLA la condición de filtrado F
        CALCULAR valores requeridos según {DF}
        EJECUTAR O

Figura 1. Estructura de una regla.

Uno de los aspectos más destacables del modelo de reglas es la posibilidad de establecer tanto la fuente como el destino de manera concreta o de manera genérica. Esto es, indicando una entidad en particular, o por contra, un tipo de entidad que denotará a todas las entidades que se instancien a partir de dicho tipo y estén en simulación. Por tanto, hay que tener en cuenta el significado de utilizar poblaciones fuente y/o destino utilizando un tipo de entidad. Si se define la fuente como un conjunto de entidades de un tipo, la regla se instanciará en caso de que cualquier entidad de dicho tipo cause el evento de tipo E. Por otra parte, si se define la población destino de la regla como un conjunto de entidades de un tipo, entonces habrá una instanciación de la regla para cada una de las entidades de la población destino. Este mecanismo es ventajoso y permite simplificar la especificación del comportamiento cuando las instancias del tipo de entidad son desconocidas a priori, o cuando existen varias instancias que deben comportarse exactamente de la misma manera.

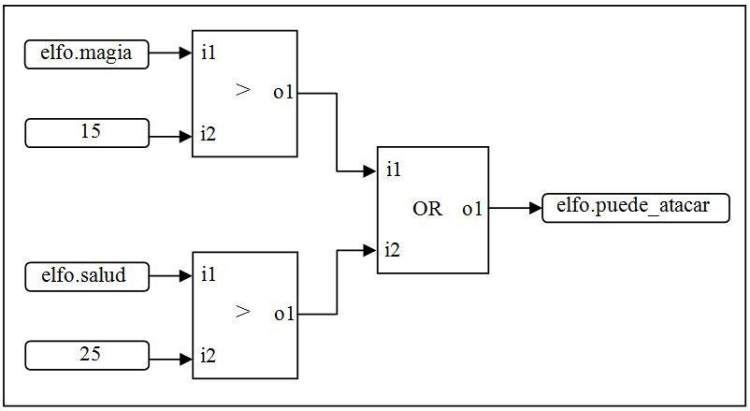
En otros sistemas para especificar el comportamiento en juegos, las reglas se especifican de manera más simple, no permitiendo por ejemplo aplicar ninguna transformación a los datos antes de ser establecidos como parámetros de la operación, y por tanto la expresividad queda limitada [15]. En el presente trabajo se ha enfatizado en proveer mecanismos visuales sencillos e intuitivos que no limiten la expresividad deseada para el lenguaje de definición de reglas propuesto. Así, en lugar de forzar a que los valores de los parámetros de una operación deban tomarse directamente de los valores que toman los atributos del evento que activó la regla, o de los valores de las propiedades de la entidad fuente, sin someterse a ningún tipo de transformación u operación, se ha optado por permitir, mediante flujos de datos, aplicar operaciones sobre los datos disponibles y combinarlos obteniendo construcciones mucho más complejas. Además, se provee un mecanismo para involucrar fácilmente propiedades de cualquier entidad que esté siendo simulada en el ecosistema, con el fin de no imponer limitaciones en el alcance de las reglas.

Plataformas generalistas para la creación de juegos como [15] soportan un potente lenguaje de scripting, pero el mecanismo visual que incluyen no permite especificar los parámetros fácilmente. Otros lenguajes dan mayor importancia a la parte visual, pero descuidan la potencia expresiva. Puesto que los modelos de flujos de datos ya han sido empleados con anterioridad como base para determinados tipos de programación visual [16][17], la propuesta que aquí se presenta se basa en combinar la sencillez de las estructuras de reglas típicas con la enorme expresividad que

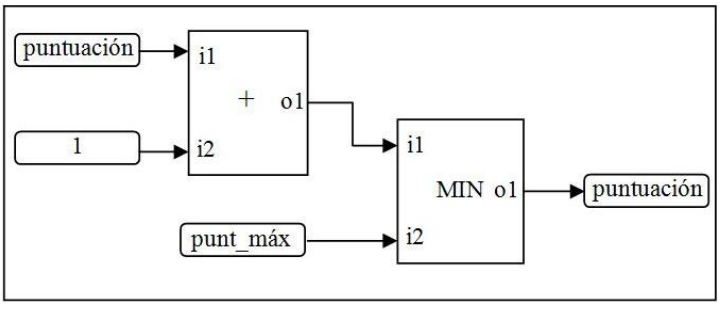
permiten las estructuras de flujos de datos, de manera que el lenguaje resultante sea más potente y usable gráficamente.

Teniendo en cuenta la formalización de las reglas propuesta anteriormente, y que deseamos ser capaces de calcular variables dentro de dicha regla, la expresividad requerida para dicho lenguaje debe al menos soportar asignaciones, operadores aritméticos y lógicos. Una gramática que describe de manera textual la expresividad buscada puede ser definida como sigue:

Instrucción := Variable '←' Expresión
Operando := Variable | Constante | Expresión | Función
Operando_parentizado := '(' Operando ')' | Operando
Operador := '+' | '-' | 'x' | 'AND' | 'OR' | 'NOT' | '<' | '>' | '=' | '<=' | '>='
Expresión := Operando_parentizado [Operador Operando_parentizado]
Función := Nombre_función '(' Parámetros ')'
Parámetros := NIL | '(' Expresión [, Expresión]* ')'
Nombre_función := 'MAX' | 'MIN' | 'ABS' | …



elfo.puede_atacar ← (elfo.magia > 15) OR (elfo.salud > 25)
(a)



puntuación ← MIN(punt_máxima, puntuación + 1)
(b)

Figura 2. Ejemplos de flujos de datos soportados por el editor. La expresión (a) muestra una condición, mientras que la (b) ilustra un cálculo de un parámetro para ser utilizado en la operación del consecuente.

La Figura 2 muestra el tipo de expresiones que deben ser soportadas por los flujos de datos. Las expresiones son como muchas otras empleadas en cálculo y matemáticas en general, mientras que visualmente tienen una estructura similar a una organización en árbol. La variable a ser calculada se encuentra en la parte derecha de la expresión, los operadores se representan como cajas con entradas y salidas, y las variables y constantes involucradas en el cálculo se conectan mediante flujos a los operadores.

Como se ha indicado anteriormente, las condiciones y las transformaciones de valores para el cálculo de parámetros y propiedades se especifica mediante flujos de datos como los ilustrados. Finalmente, la Figura 3 muestra dos ejemplos de reglas que podrían definirse en el editor propuesto para juegos reactivos. La regla (a) serviría para especificar el comportamiento de una entidad concreta en un juego de plataformas, mientras que la regla (b) define el comportamiento típico de un juego de guerra. Esta representación en lenguaje natural es tan sólo una representación textual de la capacidad expresiva que el metamodelo implementado permite.

SI/CUANDO robot1 LANZA una ocurrencia de evento de tipo Atravesar_Portal ENTONCES
    PARA CADA entidad en {robot1}
        CALCULAR robot1.posición ← Atravesar_Portal.posición_salida
        EJECUTAR Asignar valor a propiedad

(a)

SI/CUANDO un avión de Clase_Avión LANZA una ocurrencia de evento de tipo Misil_Lanzado Y SE SATISFACE la condición Misil_Lanzado.lugar = ”Tierra” ENTONCES
    PARA CADA entidad tanque en Clase_Tanque QUE CUMPLA distancia(tanque.posición,Misil_Lanzado.posición_colisión) < 10)
        CALCULAR tanque.defensa ← tanque.defensa - Misil_Lanzado.daño
        EJECUTAR Asignar valor a propiedad

(b)

Figura 3. Ejemplos textuales de reglas soportadas por el editor.

4 Evaluación sobre Flujos de Datos

Para explorar la idoneidad de la utilización de un lenguaje de flujos de datos como el que se va a utilizar en las reglas definidas en DaFRule, se llevó a cabo un pequeño estudio preliminar acerca de la comprensibilidad del lenguaje visual de flujos de datos. El estudio involucró a alumnos de 1º de Bachiller (de 16 años de edad en promedio) que estaban cursando la asignatura de Informática. Estos alumnos precisamente iban a enfrentarse por primera vez a los conceptos de programación, por lo que los resultados darían una idea acerca de la facilidad para comprender y emplear este tipo de abstracciones y estructuras que representan conceptos computacionales.

De acuerdo al currículum de la asignatura, el profesor de la asignatura de Informática dedicó 4 sesiones a enseñar los conceptos básicos necesarios (variables, operadores, instrucciones de asignación, etc.). Para la enseñanza de estos conceptos se empleó un lenguaje textual que se corresponde con el de la gramática anteriormente presentada, y el lenguaje visual para flujos de datos equivalente, siguiendo una representación como en los ejemplos de la Figura 2. Ambas aproximaciones fueron impartidas dándoles la misma importancia, y sin dar preferencia a una u otra, y fueron debidamente ejemplificadas por el profesor. Así mismo, los alumnos resolvieron algunos ejercicios conjuntamente.

Los alumnos realizaron una prueba con ejercicios relacionados con el uso de los lenguajes textual y visual vistos. Tras la prueba, solamente 24 alumnos decidieron responder al cuestionario sobre preferencias de uso que en este trabajo se reporta. En dicho cuestionario, los alumnos tuvieron que elegir, para cada cuestión, qué versión del lenguaje, textual o visual, preferían en relación a la sentencia que se les planteaba. En lugar de utilizar cuestiones con escalas Likert para valorar el grado de acuerdo y desacuerdo con la sentencia, se optó por formular las cuestiones de tal forma que tuvieran que elegir forzosamente entre textual y visual, siguiendo recomendaciones. Esta decisión fue tomada durante una reunión con los profesores, quienes sugirieron como mejor opción emplear una selección en lugar de escalas, ya que permitiría obtener una respuesta más directa de los alumnos aun en el caso de que el sujeto no tuviera su capacidad crítica muy desarrollada, como suele ocurrir en esas edades. De esta forma, se pretendió valorar si un lenguaje visual para flujos de datos parecía aportar alguna ventaja en cuanto a comprensibilidad frente al lenguaje textual, que se asemeja a las expresiones matemáticas que ya habían visto en las clases de matemáticas y que ahora estaban expandiendo con algunos conceptos computacionales.

La Tabla 1 resume los conteos del cuestionario. Es destacable que existe una notable mayor preferencia por el lenguaje visual en cuanto a la facilidad de uso, la comprensión, el aprendizaje y la utilización para cómputo, aunque para la escritura de expresiones no muestra tanta ventaja. Por tanto, a la vista de los resultados, parece realmente prometedor el uso de flujos de datos para no-programadores, proporcionando una herramienta valiosa a tener en cuenta, y es por ello por lo que definitivamente se optó por incluirlas en las reglas como se propone en el presente trabajo. Además, se espera que la retroalimentación visual del editor de reglas realmente ayude a la escritura de expresiones.

Tabla 1. Conteo para las cuestiones sobre preferencia de uso de los lenguajes.

Cuestión	Textual	Visual
Me ha resultado más fácil aprender el uso del lenguaje…	7	16
Entiendo mejor las expresiones escritas en el lenguaje…	7	15
Los diferentes elementos que componen el lenguaje visual (operadores, variables, etc.) los entiendo mejor en el lenguaje…	5	18
Aprendí más rápido el lenguaje…	6	17
Encuentro más fácil calcular los valores en el lenguaje…	4	19
Escribo más fácilmente expresiones en el lenguaje…	10	12

5 Prototipo de Pruebas para DaFRule

Se ha desarrollado un prototipo de un editor de reglas que permita validar las ideas del modelo de DaFRule. Éste ha sido desarrollado en C# y su interfaz de usuario está basada en controles y ventanas. No obstante se ha dado importancia a la distribución espacial que requiere la interacción colectiva, y se ha utilizado un HP TouchSmart IQ522, dotado de pantalla táctil de 22', para hacer las pruebas de concepto. De esta forma, ha resultado más sencillo probar la validez del modelo de reglas, además de valorar consideraciones de diseño que necesitaremos tener en cuenta para el desarrollo del editor definitivo en una superficie interactiva, como son la disposición de los elementos correspondientes a la estructura de la regla y la necesidad de disponer de un mecanismo de vistas para dar la capacidad de mostrar y editar un único flujo de datos a la vez. En esta sección se describe cómo se ha llegado a este prototipo y cómo deben editarse las reglas en él. El primer punto a resolver es cómo representar los conceptos que involucra una regla para que sean intuitivos. Para ello, la disposición de los elementos en la interfaz también juega un papel fundamental a la hora de establecer un orden en la edición, y de separar el antecedente de la regla del consecuente. Por ello, se ha optado por mostrar el antecedente de la regla en la parte izquierda de la pantalla, de manera semejante a como aparece en las proposiciones lógicas. El consecuente, por tanto, se muestra en la parte derecha, y los operadores a aplicar a los flujos de datos se encuentran entre el antecedente y el consecuente. De este modo, la secuencia de operaciones aplicadas a los datos del antecedente se producen de izquierda a derecha, confluyendo en el consecuente. Así se permite una visión clara del orden en el que se aplicarán estas operaciones, y se pueden detectar errores en dicho orden de aplicación más fácilmente.

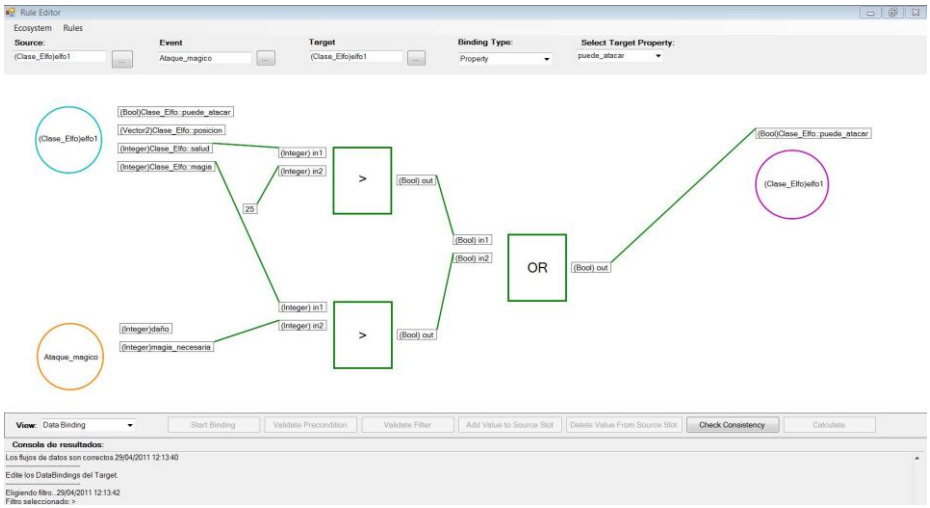


Figura 4. Definición de una regla con el prototipo del editor. La regla consta de 3 operadores, y asigna valor booleano a una propiedad de la población destino.

Por otra parte, la selección de la fuente, el evento, la población destino y el tipo de operación a ejecutar deben poder realizarse en cualquier momento de la edición, y se debe permitir cambiar de criterio también en cualquier punto, remodelando por tanto cualquier flujo de datos implicado. Estos elementos son representados en el prototipo mediante áreas circulares coloreadas de forma específica, y su selección se realiza explorando los elementos de la colección deseada. En el momento en que se tengan seleccionados correctamente, se podrán editar los flujos de datos correspondientes. Por razones de usabilidad sólo se permite editar un flujo de datos a la vez, por lo que es necesario cambiar la vista de edición seleccionando el flujo de datos deseado en un menú desplegable. Por ello, la regla completa no se muestra mientras se está editando, sino que se compone de un conjunto de vistas en función de los flujos de datos que la forman. Se requieren como mínimo dos vistas, una para la condición del antecedente y otra para la condición de filtrado. Se añadirán más vistas en función de la operación a llevar a cabo. De este modo, solo la fuente, el evento y el destino se muestran y son modificables en todas las vistas.

Los operadores se seleccionan también mediante la exploración de una colección de operadores predefinidos en el sistema, y se representan como cajas con entradas y salidas. Tanto las propiedades de la fuente y del evento, como las entradas y salidas de los operadores, se pueden unir mediante los dedos para crear los flujos de datos que definen cómo se realiza el cálculo del dato objetivo de la regla. De no ser correcta la especificación de la regla, se pueden borrar y reeditar flujos de datos según se desee. Se considera que una regla está correctamente definida cuando todos los flujos de datos, tanto del destino como de las condiciones, son correctos. La Figura 4 mostraría una regla correctamente editada en el prototipo, y la Figura 5 la exploración de los operadores para realizar la selección deseada. Ya que este prototipo es una herramienta para validar la viabilidad del modelo de reglas, se ha optado por incorporar explícitamente un mecanismo de comprobación de tipos off-line. Mediante una consola se reportan los errores encontrados en la definición de la regla referentes a los tipos de los elementos involucrados o a las conexiones establecidas. Así, mediante códigos de colores se puede mostrar al usuario aquellos flujos de datos en los que los tipos no corresponden, o flujos de datos incompletos, con el objetivo de facilitar la edición y reducir la tasa de errores en la definición a medida que el usuario progresa en su edición.

Este prototipo ha permitido identificar los requisitos de diseño necesarios para la posterior implementación de la versión definitiva del editor sobre una superficie interactiva. En primer lugar, se desea que la edición pueda ser colaborativa, ya que estos dispositivos lo permiten. Así, varios usuarios pueden cooperar editando cada uno una parte de la regla simultáneamente. Además, la edición debe realizarse totalmente con las manos y mediante el uso de elementos tangibles, para explotar todas las posibilidades de interacción que estos elementos ofrecen.

Por otro lado, los elementos de la regla, a saber fuente, destino y evento, claramente deben ser visibles. También deben poder desplazarse por el área de edición según convenga por cuestiones de espacio, ya que si la regla a editar es bastante compleja, quizá una reestructuración de la disposición de los elementos haga más clara su definición. Además, si estos elementos son también reorientables, se está facilitando la edición colaborativa, de manera que todos los usuarios inmersos en la edición pueden acceder convenientemente a ellos. En la misma línea de la

distribución espacial, tan solo se debe permitir editar y visualizar un flujo de datos simultáneamente, con el fin de hacer usable la interfaz. Otro punto a tratar sería poder incorporar al área de edición tan sólo las propiedades y atributos que se deseen para la edición de la regla, en lugar de tener todas las propiedades de la fuente y todos los atributos del evento. Esto facilitaría el aprovechamiento del espacio de edición, ya que lo más frecuente es que al definir una regla no se empleen todos estos datos, sino tan sólo un conjunto de los mismos. Además, al igual que se plantea para fuente, destino y evento, también sería deseable que estos elementos puedan situarse en cualquier punto del área de edición.

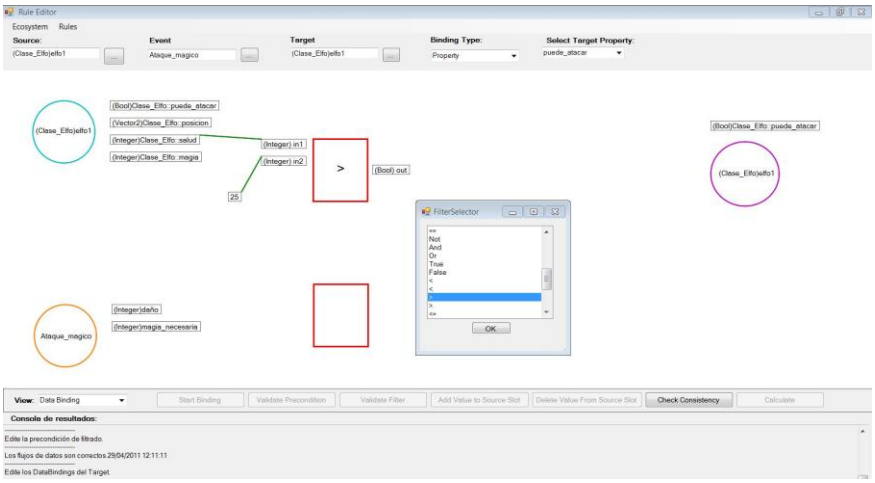


Figura 5. Selección de un operador con el prototipo del editor.

Otra cuestión importante es que para la versión definitiva del editor el mecanismo de comprobación de tipos debe ser on-line. Así, en lugar de emplear una consola y botones, durante la edición se irán reportando los errores al usuario con el mecanismo de códigos de colores propuesto en el prototipo.

En cuanto a los controles para la exploración de colecciones, se emplearán los controles multiusuario necesarios. La funcionalidad debe ser la misma, pero adaptada convenientemente a interfaces 360º para soportar la exploración de las colecciones de manera concurrente por varios usuarios situados en diferentes puntos del área de edición y con diferentes orientaciones.

Para validar que las reglas editadas con este prototipo y conforme al modelo propuesto permiten calcular correctamente los valores objetivo durante la simulación, se ha implementado un depurador. Este depurador permite cargar una definición de ecosistema que incluya las reglas editadas con el prototipo anterior, e inyectar una serie de ocurrencias de evento. Paso a paso se permite comprobar el valor de las propiedades de cualquier entidad de la simulación, así como visualizar una traza de salida que indica qué definición de regla se ha instanciado, qué entidades se han visto afectadas y de qué manera (asignación de valor a una propiedad, o invocación de un método de dicha entidad). A su vez, este depurador ha permitido validar que el cálculo del consecuente de la regla se realiza correctamente conforme a la especificación de los flujos de datos de la regla, y que por tanto el modelo propuesto

es válido y correcto, permitiendo su integración en un entorno de simulación de juegos. La Figura 6 y Figura 7 muestran la funcionalidad básica que ofrece el depurador, donde se permite comprobar el estado de las variables tras la carga de un fichero de texto que contiene un ecosistema con reglas y ocurrencias de evento, conforme se van lanzando dichas ocurrencias.

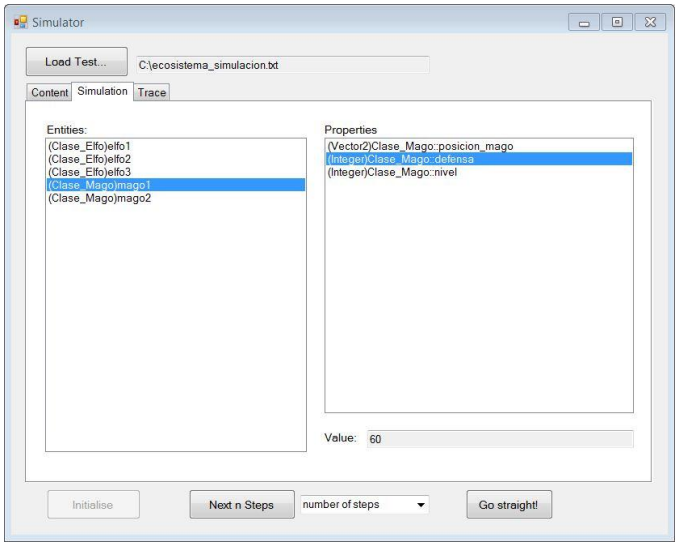


Figura 6. Visualización de propiedades de las entidades de la simulación con el depurador implementado, sin haber lanzado todavía ninguna ocurrencia de evento.

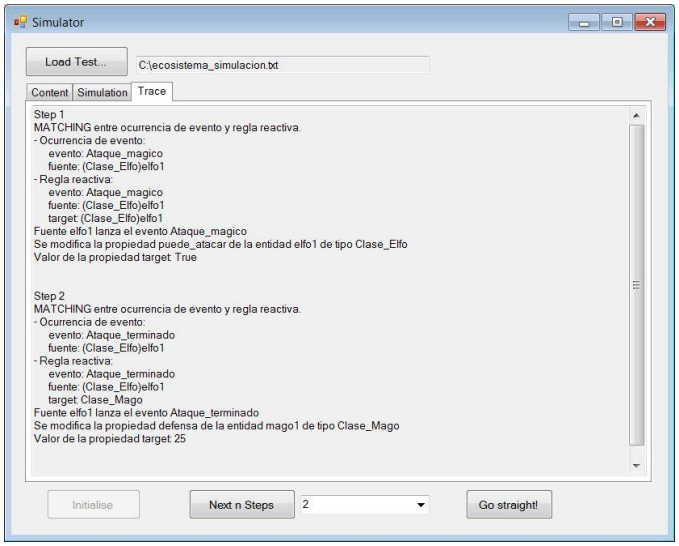


Figura 7. Traza de ejecución mostrada por el depurador tras lanzar 2 ocurrencias de evento. Se indican las reglas instanciadas, las poblaciones afectadas, la operación realizada y su resultado.

6 Conclusión

En este artículo se han presentado los fundamentos de un lenguaje de reglas que combina el uso de flujos de datos para proporcionar mayor expresividad. El interés del uso de flujos de datos viene soportado por un estudio previo con estudiantes de 1º de Bachillerato no-programadores que mostró la idoneidad de la representación visual especialmente para su compresión. De esta forma se pretende, en el contexto de la especificación de comportamiento reactivo para entidades virtuales en un juego por parte de usuarios jóvenes no-programadores, diseñar e implementar un editor de reglas con alta expresividad sin la necesidad de recurrir a la escritura de código. En relación al lenguaje se han implementado dos herramientas, el editor y el depurador, que han permitido validar el modelo de definición de reglas propuesto, y por tanto abrir nuevas líneas de trabajo destinadas a dotar a este lenguaje visual de una mayor usabilidad y un mayor campo de posibilidades de interacción. Además, dichas herramientas han permitido identificar los requisitos de diseño descritos previamente, necesarios para la implementación de una versión tangible.

Como se comentó en la introducción, las ITUs como las superficies interactivas permiten posibilidades nuevas de interacción deseables para el sistema propuesto en este trabajo. La aplicación de métodos constructivos y la combinación correcta de las interfaces apropiadas junto con tecnologías interactivas tales como los tableros interactivos, fomentaría claramente el aprendizaje creativo mediante el desarrollo de tres factores considerados como necesarios: conocimiento, pensamiento creativo y motivación. Un entorno de aprendizaje basado en superficies tangibles interactivas orientado a la creación y simulación de ecosistemas virtuales resulta prometedor, ya que combina aspectos que no han sido considerados en enfoques anteriores, tales como la cooperación, colaboración y comunicación cara a cara, y el aprendizaje basado en experiencias.

Por tanto, nuestro trabajo futuro más inmediato es desarrollar un editor de reglas como el descrito e integrar nuestro animador de reglas con un simulador de ecosistemas virtuales para una superficie interactiva. Esto permitirá desarrollar todos los requisitos identificados con el prototipo, además de borrar e insertar elementos con mayor facilidad, e interactuar con cualquier elemento involucrado en la edición desplazándolo o rotándolo de manera natural. Otra de las ventajas será poder emplear controles 360º diseñados específicamente para tal fin, así como exploradores de colecciones más adecuados para nuestro propósito explotando al máximo las capacidades cooperativas que brinda esta tecnología.

Agradecimientos

El presente trabajo ha sido financiado por el Ministerio de Ciencia e Innovación cómo parte del proyecto TIN2010-20488 del Plan Nacional de Investigación Fundamental no Orientada. Nuestro agradecimiento al Col·legi Parroquial D. José Lluch de Alboraya, especialmente a los profesores y estudiantes que participaron en el estudio empírico que se describe en este artículo. A. Catala es beneficiario de una beca FPU del Ministerio de Educación con referencia AP2006-00181.

Referencias

1. Wing, J.M.: Computational thinking. Commun. ACM vol.49, no.3, pp.33--35(Marzo 2006).
2. Kelleher, C., Pausch, R.: Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. ACM Comput. Surv. 37, 2, pp. 83--137 (Junio 2005).
3. Hornecker, E., Buur, J.: Getting a Grip on Tangible Interaction: A Framework on Physical Space and Social Interaction. Proceedings CHI, pp. 437--446 (2006).
4. Marshall, P., Rogers, Y., Hornecker, E.: Are Tangible Interfaces Really Any Better Than Other Kinds of Interfaces? Workshop on Tangible User Interfaces in Context and Theory, ACM CHI (2007).
5. Carroll, J.M.: Becoming social: Expanding scenario-based approaches in HCI. Behaviour and Information Technology, 15(4), pp. 266--275 (1996).
6. Pane, J.F., Ratanamahatana, C.A., Myers, B.A.: Studying the Language and Structure in Non-Programmers Solutions to Programming Problems, International Journal of Human-Computer Studies, vol. 54, no. 2, pp. 237--264 (Febrero 2001).
7. Good, J., Howland, K., Nicholson, K.: Young People's Descriptions of Computational Rules in Role-Playing Games: An Empirical Study, Visual Languages and Human-Centric Computing (VL/HCC), 2010 IEEE Symposium, pp. 67--74 (21-25 Sept. 2010).
8. Resnick, M., et al.: Scratch: programming for all. Commun. ACM 52, 11, pp. 60--67 (Noviembre 2009).
9. Begel, A.: LogoBlocks: A graphical programming language for interacting with the world. Electrical Engineering and Computer Science Department. MIT, Cambridge, MA (1996).
10. Kelleher, C., Pausch, R.: Using storytelling to motivate programming. Commun. ACM 50, 7, pp. 58--64 (Julio 2007).
11. Repenning, A., Ioannidou, A., Zola, J.: AgentSheets: End-User Programmable Simulations. Journal of Artificial Societies and Social Simulation, 3(3) (2000).
12. Gallardo, D., Julia, C.F., Jorda, S.: TurTan: A tangible programming language for creative exploration, In proc. Tabletops, pp. 89--92 (2008).
13. Suzuki, H., Kato, H.: Interaction-level support for collaborative learning: AlgoBlock an open programming language. Int. conf. on Computer support for collaborative learning, pp. 349--355 (CSCL '95).
14. Horn, M.S., Jacob, R. J. K.: Designing tangible programming languages for classroom use. TEI '07. ACM, New York, NY, USA, 159--162 (2007).
15. Game Maker Website: http://www.yoyogames.com/gamemaker/.
16. Blanchard, C., et al.: Reality built for two: a virtual reality tool. In Proceedings of the 1990 symposium on Interactive 3D graphics. ACM, New York, NY, USA, 35--36 (I3D '90).
17. Hancock, C.: Children's Understanding of Process in the Construction of Robot Behaviors. In Proceedings of Symposium on Varieties of Programming Experiences (2001).