Un entorno de aprendizaje y una propuesta de enseñanza de Simulación de Eventos Discretos con GPSS

LIC. GONZALO LUJÁN VILLARREAL

Director: ING. MARISA RAQUEL DE GIUSTI
Co-Director: DRA. SILVIA GORDILLO

Facultad de Informática - Universidad Nacional de La Plata

Tesis presentada para obtener el grado de Doctor en Ciencias Informáticas

Mayo 2013

Índice

Dedicado a mis viejos que, desde sus realidades cotidianas, me demuestran día a día que el esfuerzo, la constancia y el trabajo honesto aseguran el alcance de los objetivos planteados, sin importar cuán ambiciosos o imposibles sean.

Agradecimientos

A mis directoras Marisa y Silvia, por la insistencia y el apoyo que me dieron durante el doctorado.

A la Facultad de Ingeniería de la UNLP, por ofrecerme el espacio físico por medio de PREBI-SEDICI, y por la excelencia del personal de la Secretaría de Investigación y Transferencia por su calidez, su predisposición y su eficiencia.

Al personal de la oficina de Posgrado de la Facultad de Informática, por su colaboración constante y la excelente atención y predisposición de las personas que allí trabajan.

A los alumnos que participaron en los encuentros experimentales. Sin sus ganas y dedicación este trabajo hubiera sido muy difícil.

A Jusmeidy Zambrano, por su participación, sus consejos y sus aportes en los encuentros, y por la dedicación y la excelencia impresa en todo su trabajo.

A Analía Pinto y a José D. Texier, por la paciencia para leer, comentar y corregir esta tesis.

A la Dra. Gisela Lamas, por sus consejos y los conocimientos brindados.

Al Lic. Ariel J. Lira, por sus consejos y aportes para mejorar las herramientas de software desarrolladas en este trabajo.

Al CONICET, por brindarme el apoyo económico que me permitió realizar este doctorado.

Resumen

La enseñanza en el área de simulación de eventos discretos requiere integrar una variedad de conceptos teóricos y ponerlos en práctica a través de la creación y ejecución de modelos abstractos de simulación, con el objetivo de recopilar información que pueda traspolarse hacia los sistemas reales. Para construir modelos, ejecutarlos y analizar los resultados de cada ejecución se utilizan herramientas de software cada vez más sofisticadas que permiten expresar los elementos de los modelos en términos de entidades abstractas y relaciones, y que recopilan gran cantidad de datos y estadísticas sobre cada una de estas entidades del modelo. GPSS es una de estas herramientas, y se compone de un lenguaje de programación por bloques y un motor de simulación que traduce estos bloques en distintas entidades del modelo. A pesar de que su primera versión data de 1961, GPSS es aún muy utilizado por profesionales y empresas, y es una de las herramientas más utilizadas para la enseñanza de simulación de eventos discretos por instituciones académicas de todo el mundo.

El avance de la capacidad de cómputo de las computadoras ha permitido incorporar una mayor cantidad de herramientas y funciones a las distintas implementaciones de GPSS. Mientras que esto representa una ventaja para sus usuarios, requiere también un cada vez mayor esfuerzo por parte de los docentes para enseñar a sus estudiantes a aprovechar todo su potencial. Muchos docentes e investigadores han buscado optimizar la enseñanza de simulación de eventos discretos desde múltiples ángulos: la organización del curso y la metodología de enseñanza, la creación de elementos de aprendizaje que ayuden a aplicar los distintos elementos teóricos, la generación de herramientas para construir modelos GPSS, y la construcción de herramientas para comprender el motor de simulación por dentro.

En esta tesis se introduce una herramienta de software que permite construir modelos GPSS de manera interactiva, cuyo diseño fue pensado para integrar los elementos teóricos del curso con los objetos y entidades de GPSS. Esta herramienta también permite ejecutar estos modelos y analizar con alto nivel de detalle su evolución a través del tiempo de simulación, lo que permite a los estudiantes comprender cómo funciona el motor de simulación y cómo interactúan las distintas entidades entre sí. Se incluye también una propuesta de enseñanza basada en una fuerte participación de los estudiantes, que, por medio de esta nueva herramienta, les permite incorporar los conceptos más fácilmente. Esta propuesta de enseñanza fue puesta a prueba con alumnos del área de sistemas, quienes tomaron un curso que contiene los mismos elementos teóricos y prácticos de un curso tradicional, pero con una organización diferente. Entre los resultados logrados se destacan una reducción del tiempo requerido para aprender los conceptos de GPSS cercana al 50%, una mayor capacidad por parte de los alumnos para asimilar conceptos y derivar nuevos conceptos por si solos, a partir de conceptos adquiridos previamente.

Capítulo 1

Introducción

La simulación por computadora es una técnica para predecir cómo los sistemas, nuevos, alterados o supuestos, se comportarán bajo ciertas condiciones. La mayoría de los sistemas del mundo real pueden ser simulados, por lo general, a partir de modelos abstractos y de alguna herramienta en particular.

Simular un sistema "real" requiere conocer profundamente dicho sistema, sus elementos y su relación con el entorno o subsistema en el cual está embebido, lo que se conoce como entradas y salidas del sistema. Todo conocimiento del sistema sirve para construir un modelo abstracto del mismo, en el cual se considerarán los aspectos más importantes de acuerdo al objetivo del estudio (el estudio del sistema debe tener un objetivo o propósito). Estos aspectos abarcan elementos del sistema, interacciones entre estos elementos, y elementos del medio o entorno del sistema. Desde luego, el modelo no puede representar la totalidad del sistema original, pues sería un modelo excesivamente complejo y difícil de manejar.

El modelo abstracto es luego puesto a prueba bajo distintas condiciones, lo cual permite observar cómo se comporta el modelo a medida que las condiciones varían, y a partir de allí extrapolar esta información hacia el sistema real. Cuanto mayor sea el número de repeticiones de estas pruebas, más exacta será la extrapolación entre el

modelo y el sistema, pues —suponiendo que el modelo esté bien realizado— el margen de error (o sea, la diferencia entre los datos que genera el modelo y lo que sucede en el sistema real) se reducirá con cada replicación de las pruebas.

En líneas generales, los conceptos arriba descritos conforman el corazón de un curso de Modelos y Simulación[1], más allá del tipo de sistemas con el que se trabaje, la tecnología que se aplique, y las herramientas que se utilicen para construir modelos y probarlos (ejecutarlos). Estos conceptos, presentados de manera simple, implican en realidad una gran diversidad de conceptos subyacentes, que son requeridos y aprendidos durante el curso de Modelos y Simulación: sistemas, modelos, estadísticas, diseño de experimentos y minería de datos son quizás los más importantes, aunque son también sólo la punta del iceberg. El diseño de modelos de simulación para distintos tipos de sistemas requiere habilidades específicas según el área de aplicación. En el contexto de este trabajo, se considerarán los sistemas de eventos discretos, también conocidos como sistemas gobernados por colas.

El estudio de simulación de sistemas de eventos discretos requiere, por un lado, conocer las principales características de este tipo de sistemas, como los distintos tipos de entidades, eventos, gestión del tiempo de simulación, sistemas de colas, servidores y servicios, entre otros, y, por otro, el dominio de al menos una herramienta tecnológica que permita construir estos modelos, ejecutarlos y obtener resultados que puedan ser analizados, comparados y optimizados. Entre la gran variedad de herramientas existentes en la actualidad para el modelado de sistemas discretos, se destaca el Sistema de Simulación de Propósito General o GPSS (General Purpose Simulation System). Este sistema, creado en 1961 por G. Gordon de los laboratorios de IBM, se compone de un

[1] Un gran número de universidades de todo el mundo incluyen al menos un curso del área de simulación en sus carreras de informática e ingeniería. Se incluyen aquí los enlaces a algunos programas de estos cursos:
Simulación de Eventos Discretos, Facultad de Ingeniería, Universidad de la República, Uruguay
Modelado y Simulación de Sistemas de Eventos Discretos, Universidad del País Vasco
Modelación y Simulación de Sistemas de Eventos Discretos, Instituto Politécnico Nacional, México
Simulación de Eventos Discretos, UBA, Argentina
Modelos y Simulación, UNLP, Argentina
TM8105 Advanced Discrete Event Simulation Methodology, Norwegian University of Science and Technology, Norway
SYS 6034: Discrete-event Stochastic Simulation, Universtity of Virgina, United States of America
System Simulation - a practical course, International University of Japan, Japan
Discrete-Event Simulation (DES) for Operations and Engineering Management, University of the West of England, United Kingdom

lenguaje de programación integrado principalmente por bloques y comandos, y un motor de simulación que traduce estas sentencias —bloques y comandos— en un conjunto de entidades que luego interactúan unas con otras, lo cual se conoce como correr o ejecutar el modelo.

Tanto las sentencias de GPSS como las entidades del motor de simulación se caracterizan por un alto nivel de generalidad y abstracción, lo cual permite al programador avezado en este sistema construir modelos extremadamente complejos con pocas líneas de código —en especial, comparando el mismo modelo construido con un lenguaje de propósito general como C++, Pascal o Java—. Las sucesivas implementaciones de GPSS han mejorado de manera considerable la eficiencia en la etapa de ejecución de los modelos, lo que ha sido también acompañado por el extraordinario incremento en la capacidad de cálculo de las computadoras personales. Gracias a ello, es posible ejecutar modelos GPSS muy extensos en cuestión de segundos, y obtener posteriormente una amplia variedad de datos que sirven para interpretar los resultados de la simulación.

Aquí radica otra de las principales características de GPSS: durante la ejecución de los modelos de simulación, el motor de simulación de GPSS recolecta de manera automática gran cantidad de datos tanto de las entidades que participan en la ejecución como de la simulación en sí, y genera, a partir de estos datos, un conjunto de estadísticas muy variadas que permiten al analista interpretar cómo ha sucedido la ejecución de la simulación y cómo se ha comportado cada una de las entidades. Además, mediante la repetición de ejecuciones —habilidad con la que cuenta la mayoría de las implementaciones modernas de GPSS—, es posible realizar experimentos que permitan identificar interacciones entre parámetros de entrada del sistema y los valores de salida, o incluso realizar optimizaciones sobre el modelo.

La amplia variedad de herramientas que posee GPSS para la construcción de modelos, su ejecución, el análisis de resultados y la realización de experimentos representan un gran potencial, pero también un importante inconveniente al momento de la enseñanza en ámbitos académicos: se trata de una herramienta de uso profesional, muy avanzada y completa, pero a la vez extremadamente abstracta y compleja de utilizar. Además, el paradigma de programación utilizado, basado en bloques y comandos, resulta poco

natural incluso para los estudiantes que ya conocen técnicas de programación modernas y lenguajes de programación de cuarta generación. Esto se debe a que los programas escritos en GPSS carecen de una estructura jerárquica clara, las entidades utilizadas en los modelos no son definidas en ningún lugar del código, y el uso de cada sentencia GPSS engloba la ejecución de un amplio conjunto de operaciones y rutinas entre distintas entidades del motor de simulación de manera automática y transparente para el programador. Además, a diferencia de lo que ocurre con la mayoría de los lenguajes de programación, aprender GPSS no significa solamente aprender los elementos sintácticos y las implicancias semánticas de utilizar dichos elementos, propio de los lenguajes de programación tradicionales. El aprendizaje de GPSS requiere también comprender un motor de simulación que funciona siguiendo una visión del mundo basada en la interacción de procesos [Pollacia89], junto a un amplio conjunto de entidades con atributos y rutinas internas. Cabe mencionar también que las implementaciones profesionales de GPSS son todas de carácter comercial, poseen licencias privativas, y sólo son compatibles con entornos MS Windows™. Si bien en algunos casos las empresas ofrecen licencias de menor costo para ámbitos académicos, o versiones reducidas en funciones para estudiantes universitarios, no existen implementaciones profesionales de GPSS de uso libre y abierto, ni tampoco existen herramientas multiplataforma o independientes de la plataforma. En algunos casos se han realizado desarrollos basados en la web, pero estos desarrollos se encuentran ya desactualizados o corren sobre tecnologías hoy en día obsoletas.

Durante los últimos 20 años han surgido numerosos proyectos de desarrollo y líneas de investigación que buscan optimizar la enseñanza en el área de simulación de eventos discretos por un lado, lidiando con la gran diversidad temática, los objetivos del curso y la disparidad de conocimientos preexistentes en los alumnos, y de enseñanza del lenguaje GPSS por el otro, mediante librerías que extienden a los lenguajes de programación tradicional, sistemas icónicos, o nuevos lenguajes de programación que facilitan el aprendizaje, principalmente en las primeras etapas del curso de Simulación de Eventos Discretos. Todos estos trabajos consideran distintos aspectos del aprendizaje de simulación de eventos discretos de manera aislada: algunos buscan implementar interfaces de trabajo más intuitivas, otros buscan acortar la distancia entre la sintaxis de GPSS y los lenguajes de programación de propósito general, otros introducen asistentes

que generan el código GPSS de manera semiautomática, algunos buscan modificar la forma en que se dicta el curso de simulación, y unos pocos buscan integrar el entorno de trabajo con otros entornos interactivos preexistentes. Sin embargo, estos trabajos no consideran los distintos aspectos como un todo: el diseño del curso de simulación, la idiosincrasia de los estudiantes, y la herramienta tecnológica que se utiliza durante el proceso de enseñanza.

El objetivo de esta tesis es, entonces, proponer una alternativa a la enseñanza tradicional de Simulación de Sistemas de Eventos Discretos. Esta propuesta se compone de una herramienta interactiva de modelado y ejecución de simulaciones en GPSS, que permite a los estudiantes generar modelos simples mediante diálogos, ayudas contextuales y elementos visuales. Estos modelos son ejecutados por un motor de simulación de GPSS adaptado con fines pedagógicos. Una de las principales características de esta adaptación es la capacidad del motor de simulación de registrar los cambios en cada una de las entidades del modelo durante la ejecución de la simulación, lo cual permite luego a los estudiantes analizar el historial de la ejecución de sus modelos, explorando cada una de las entidades paso a paso, analizando cómo las distintas entidades han interactuado unas con otras, y comprendiendo finalmente cómo y porqué se han logrado estos resultados en la simulación. Para esto, la herramienta propuesta —denominada GPSS Interactivo o GPSSi— presenta los resultados de la ejecución de la simulación no como un reporte de texto plano (como suelen realizar las implementaciones profesionales de GPSS), sino como varios conjuntos de entidades que componen el modelo. Cada una de estas entidades puede ser explorada por dentro, y el usuario puede avanzar o retroceder el reloj interno de la simulación, a fin de comprender qué sucedía en instantes anteriores o posteriores. De este modo, la herramienta permite explicar de manera visual muchos de los conceptos teóricos vistos durante las clases, como los sistemas de colas, los elementos de las distintas entidades, y el funcionamiento del motor de simulación.

Por otra parte, la propuesta incluye un rediseño del curso de Modelos y Simulación, pensado específicamente para aprovechar los distintos elementos integrados en GPSS Interactivo, pero también para lograr un aprendizaje más significativo de los distintos conceptos que se brindan a lo largo del curso. Este rediseño del curso abarca no sólo una

reorganización temática, sino también un cambio de rol tanto del docente como de los alumnos. Los alumnos adoptan un rol más participativo y el docente fomenta esta participación en todo momento. De este modo, muchos conceptos teóricos son descubiertos por los propios alumnos, con la guía del docente, apoyándose en conceptos previamente adquiridos.

Este nuevo diseño del curso, junto a la herramienta GPSS Interactivo, fue puesta en práctica en el marco del curso de Modelos y Simulación de la Facultad de Informática de la Universidad Nacional de La Plata. Se seleccionó un grupo de alumnos, con quienes se trabajó bajo este nuevo esquema, utilizando esta nueva herramienta como principal soporte para la enseñanza. Si bien los conceptos teóricos fueron esencialmente los mismos, el orden de dictado fue distinto. De los distintos encuentros se obtuvo mucha información valiosa, lo cual permitió ajustar los siguientes encuentros y las actividades a realizar con los alumnos. Entre los resultados alcanzados, se observó que los alumnos habían logrado adquirir la capacidad de interpretar y generar por sí solos los nuevos conceptos teóricos, que el nivel de aprendizaje había resultado muy similar al del curso tradicional de Modelos y Simulación, y que el tiempo requerido para impartir los mismos temas se había reducido considerablemente.

Organización de la tesis

En el capítulo 2 de esta tesis se realiza una revisión de la bibliografía relativa a la enseñanza de simulación de eventos discretos, a los entornos de enseñanza interactivos, y a los desarrollos orientados a mejorar la enseñanza del lenguaje GPSS. En la última sección de este capítulo se incluye una revisión de las principales implementaciones comerciales y profesionales.

En el capítulo 3 se incluye una descripción general de los conceptos que se dictan en el curso de Modelos y Simulación. Aquí se incluyen aspectos de sistemas, modelos, la simulación como herramienta, tipos de simulaciones, y tecnologías para realizar simulaciones. Este capítulo incluye también un análisis de los principales lenguajes de programación de simulación de eventos discretos, y un repaso de los conceptos básicos de GPSS: bloques, comandos, sentencias, mecanismo de planificación de transacciones, etc.

El capítulo 4 explica en detalle la herramienta GPSS Interactivo. Esto incluye la arquitectura general de la aplicación, las distintas aplicaciones que la componen, las tecnologías y herramientas con las que fueron desarrolladas e integradas, y los principales módulos de cada una de ellas. En este capítulo se explican también los distintos aspectos de diseño que se han incluido en las distintas aplicaciones con fines didácticos.

En el capítulo 5 se describe en qué consiste el rediseño del curso de Modelos y Simulación. En este capítulo también se explica cómo se realizó la experimentación con los alumnos, es decir, cómo se aplicó este nuevo curso, junto a GPSS Interactivo, en el marco de un curso real de Modelos y Simulación, con alumnos reales. La última parte de este capítulo contiene un análisis pormenorizado de los resultados obtenidos, tanto durante cada uno de los encuentros como al finalizar el curso.

En el capítulo 6 se realizan algunas conclusiones generales de este trabajo. Este capítulo incluye también líneas de trabajo futuro, tanto relativas al desarrollo de GPSS Interactivo como también al curso de Modelos y Simulación.

Finalmente, en el capítulo 7 se listan las referencias bibliográficas mencionadas a lo largo de la tesis.

Capítulo 2

Estado del arte

La simulación de eventos discretos soportada por computadoras es una técnica para el análisis y estudio de ciertos tipos de sistemas que ya tiene más de 50 años, y que aún continúa siendo vigente y muy utilizada. Esta técnica ha sido enriquecida con herramientas diseñadas para construir modelos, ejecutarlos y analizar los resultados; las herramientas se han vuelto cada vez más sofisticadas y completas, acompañando el crecimiento exponencial en la capacidad de cálculo y almacenamiento de las computadoras. Un caso concreto de este avance es el lenguaje de simulación de propósito general GPSS, utilizado tanto por profesionales como en el ámbito académico.

La simulación de eventos discretos ha cobrado cada vez más relevancia en el ámbito profesional de un creciente número de disciplinas que estudian este tipo de sistemas: las distintas ramas de la ingeniería, ciencias de la computación y ciencias económicas son algunos de los ámbitos donde se utiliza esta técnica [Ståhl00]. Por este motivo, cada más facultades[2] incluyen cursos de simulación en general, simulación de eventos discretos, diseño de experimentos, entre otros.

Para hacer uso correcto de esta técnica es necesario contar con un profundo conocimiento sobre las herramientas informáticas que permiten simular sistemas discretos. Sin embargo, esto presenta enormes desafíos en el ámbito académico, puesto que las herramientas profesionales son cada vez más complejas, los avances en investigación y desarrollo requieren que cada vez se incluyan más temas en los cursos de

[2] En el capítulo 1 de esta tesis se mostraron algunos ejemplos de carreras de sistemas e ingeniería de universidades de todo el mundo que incluyen cursos de simulación en su currícula.

simulación, pero la duración de los cursos debe mantenerse acotada a uno o dos semestres. Por este motivo, en los últimos 20 años se han generado muchos proyectos de investigación con el objetivo de optimizar la enseñanza de simulación, mediante el diseño de entornos interactivos de enseñanza, la implementación de herramientas de simulación específicas para entornos de aprendizaje, y el diseño de marcos de trabajo para cursos de simulación. En este capítulo se ha realizado una revisión de muchos de estos trabajos, sus propuestas tanto desde el punto de vista tecnológico como pedagógico, sus ventajas y desventajas, y el aporte que han realizado en esta tesis.

Entornos de aprendizaje interactivo y enseñanza de la simulación

Con el rápido desarrollo de nuevas tecnologías, los entornos de aprendizaje interactivo se encuentran cada vez más difundidos, se están aplicando a ámbitos muy diversos y están influenciando profundamente la práctica diaria de educación [Dillon98]. Los distintos tipos de entornos de aprendizaje interactivos (Interactive Learning Environment, ILE) ofrecen diversos tipos de asistencia y soporte para los estudiantes, que se traducen en el diseño de la interfaz del usuario, la disposición de los objetos en pantalla y la interacción entre el usuario y dichos objetos, los textos, íconos e imágenes, y los sistemas de ayuda implícitos y explícitos. Al momento de diseñar un ILE, es preciso considerar cómo esta herramienta será utilizada por los estudiantes y, especialmente, cómo reaccionarán los mismos cuando requieran algún tipo de asistencia o ayuda, ya sea por parte del docente o por parte del sistema. En la mayoría de los casos, los ILE proveen ayuda bajo demanda, que abarca desde consejos específicos según el contexto [Anderson95] a material enlazado especialmente diseñado, libros de texto y glosarios en línea [Aleven00]. En muchos casos, también se incluye información contextual o icónica siempre visible, que funciona como sistema de ayuda implícito para los estudiantes.

Además, al momento de diseñar un ILE, se deben considerar no sólo las funciones de interacción entre los usuarios y el sistema, sino también la forma en que estos sistemas asisten a los estudiantes a la hora de requerir ayuda, ya sea implícita o explícita. Los ILE serán mucho más valiosos si se encuentran maneras de diseñarlos para que ayuden a los estudiantes a convertirse en mejores buscadores de ayuda [Aleven01].

Vincent Aleven *et al.* [Aleven03] realizaron un amplio estudio sobre cómo los

estudiantes buscan ayuda durante el proceso de aprendizaje, y cómo se debe diseñar esta ayuda en el contexto de los ILE. En su artículo, los autores sugieren los principales motivos que explican el comportamiento de búsqueda de ayuda inefectiva desde una perspectiva psicológica. Mencionan en su trabajo el boceto de un modelo, que ayuda a comprender el proceso de búsqueda de ayuda de los estudiantes [Aleven01A]. Este modelo, diseñado originalmente para el contexto del aula y adaptado para los ILE —en particular, para el Tutor Cognitivo de Geometría— consiste de cinco etapas secuenciales que atraviesan los estudiantes durante el desarrollo de alguna actividad o tarea:

1. El estudiante se da cuenta de que necesita ayuda, o sea que la tarea a realizar es muy difícil y/o que no puede continuar avanzando. En el contexto de los ILE, el *feedback* del sistema que recibe el alumno es esencial para alcanzar esta etapa.

2. El estudiante decide buscar ayuda, lo cual implica que debe considerar toda la información disponible y decidir si realmente precisa ayuda. Por lo general, los estudiantes se preguntan primero a sí mismos, buscando la respuesta o solución correcta para la tarea antes de decidir si deben pedir ayuda. Algunos factores propios del contexto del aula dejan de tener sentido al trabajar con ILE: por ejemplo, el miedo a ser visto como incompetente ya sea por sus pares como por el docente. Otros factores pueden surgir específicamente con los ILE: algunas herramientas penalizan a los estudiantes con el uso de la ayuda, descontando puntos al finalizar la tarea.

3. El estudiante debe identificar un ayudante potencial. En en aula, este rol lo puede cumplir tanto el docente como un par (otro estudiante), pero en entornos interactivos no siempre está claro quién (o qué) puede proveer ayuda, y los estudiantes pueden tener muchas más opciones que en un contexto social; incluso si la herramienta cuenta con funciones de ayuda, el estudiante aún puede decidir preguntarle al docente, consultar a sus compañeros o buscar en un libro. Además, los ILE pueden tener más de un método de ayuda: por ejemplo, consejos contextuales y un glosario.

4. El estudiante debe ser capaz de expresar la solicitud de ayuda en forma adecuada, lo que suele estar mediado por el conocimiento y las habilidades de cada estudiante. Al utilizar ILE, el estudiante tiene por lo general menos flexibilidad para expresar una solicitud de ayuda. En los sistemas de ayuda que no proveen información sensible al contexto, el estudiante puede tener que realizar más trabajo para encontrar información

relevante y juzgar su aplicabilidad al problema en cuestión.

5. Finalmente, el estudiante debe evaluar el episodio de búsqueda de ayuda: una vez que ha recibido ayuda, debe decidir en qué grado dicha ayuda le ha servido para alcanzar sus objetivos. Si no ha servido, deberá volver a solicitar ayuda. En este paso, el estudiante debe integrar la nueva información con su conocimiento preexistente y evaluar la calidad de la ayuda que ha recibido. La inmediatez de los ILE para responder las solicitudes de ayuda puede reducir su motivación para evaluar el episodio de búsqueda de ayuda. La respuesta que entrega el ILE suele hacer que los alumnos puedan decidir rápidamente si pueden proseguir con la tarea luego del episodio de ayuda.

Dadas estas diferencias arriba mencionadas, no se debe asumir automáticamente que los análisis y descubrimientos sobre búsqueda de ayuda en contextos sociales o en el aula pueden trasladarse directamente al contexto de los ILE.

Existen muchos factores que influyen en la forma de buscar ayuda de los alumnos. Varios estudios [Wood99][Renkl02][Puustinen98] indican que los estudiantes con menor conocimiento previo —y, en particular, aquellos que necesitan más ayuda— tienen menos probabilidades de utilizar apropiadamente los sistemas de ayuda, especialmente cuando la ayuda está bajo el control de los estudiantes. Sin embargo, los autores recomiendan ceder el control del sistema de ayuda a manos de los estudiantes de manera paulatina, ya que esto fomenta su capacidad de autodeterminación, de analizar información ofrecida por el ILE, y de mejorar por sí mismos el proceso de búsqueda de ayuda. Esto requiere, por supuesto, asegurar que la ayuda se utilice de manera adecuada y que esté disponible en el momento indicado.

Algunos estudios indican que no es posible diseñar un sistema de ayuda general aplicable a todos los casos, pues diferentes tipos de estudiantes necesitan diferentes tipos de ayuda [Arrollo00, Aleven03]. Por este motivo, el diseño de los sistemas de ayuda debe también considerar las diferencias entre las distintas clases de usuarios. Por ejemplo, se ha observado que el grado de abstracción de la ayuda interactúa con el desarrollo cognitivo: los estudiantes con mayor desarrollo cognitivo aprenden mejor cuando reciben consejos (*hints*) contextuales abstractos, y los estudiantes con menor desarrollo cognitivo se benefician más de los consejos concretos [Arrollo00]. Esto indica que en ciertos casos el sistema debe proveer ayudas y consejos contextuales más

evidentes y puntuales, mientras que en otros casos es conveniente el uso de ayudas más abstractas y menos evidentes. Las ayudas concretas pueden brindarse de manera directa —sin que el estudiante las requiera— en ciertas áreas del ILE, mientras que las ayudas abstractas pueden mostrarse a demanda, o sea cuando el estudiante decide que necesita más información para cumplir la tarea. De este modo, se simplifica la tarea de los estudiantes con menor desarrollo cognitivo, dejando el control de la ayuda a cargo del sistema, y se fomenta la capacidad de búsqueda de ayuda y análisis de la misma a medida que aumentan las habilidades de los estudiantes, otorgándoles el control del sistema de ayuda.

En resumen, el sistema de ayuda es una parte fundamental del ILE, y se debe considerar cómo se integrará con las actividades que realizarán los alumnos, en qué momento lo hará y de qué manera. En el diseño del sistema de ayuda entran en juego muchos factores: el desarrollo cognitivo de los alumnos, si la ayuda es global o sensible al contexto, la carga cognitiva de la ayuda, el uso de diferentes representaciones y el nivel de abstracción, y finalmente en manos de quién recae el control el sistema de ayuda: el estudiante o el ILE [Arrollo01].

Como se dijo previamente, el diseño de un ILE debe también considerar el ámbito en que se utilizará, el momento de aplicación, y la metodología de enseñanza que se seguirá a lo largo del curso. En el caso de la simulación de eventos discretos en general, y de GPSS en particular, existen algunos trabajos de investigación en los que se proponen varios elementos que deben considerarse a la hora de preparar un curso para esta área, ciertas características y requisitos que deben cumplir las herramientas de software utilizadas para la enseñanza, y finalmente algunas implementaciones de ILE diseñados específicamente para la enseñanza de GPSS y simulación de eventos discretos.

En el año 2009, Heriberto García y Martha A. Centeno, del Departamento de Ingeniería Industrial y de Sistemas del Tecnológico de Monterrey (México), propusieron un *framework* para el diseño de cursos de simulación de eventos discretos [García09]. En dicho trabajo, los autores mencionan una característica común de los cursos de Simulación de Eventos Discretos (Discrete Event Simulation, DES) en muchas instituciones: los alumnos ya han tomado un curso de estadística, el cual suele tener muchos estudiantes y por ende tiende a ser más abarcativo que profundo. A diferencia

de esto, un curso de DES requiere un profundo conocimiento de los temas que se imparten en los cursos de estadísticas (análisis estadístico, modelado estocástico, y análisis de sistemas). Se ha observado también que, en algunos casos, incluso los estudiantes no han cursado estadística aún. Esto hace que los docentes deban, o bien dedicar tiempo a recordar estos conceptos, o bien impartirlos de cero. Dada esta variabilidad, antes de comenzar el curso el docente debe identificar los conceptos a enseñar, el orden y los métodos de enseñanza.

Por otro lado, los cursos de DES deben generar en los estudiantes un aprendizaje profundo y significativo [Palmero04], y alcanzar un alto nivel de comprensión de los conceptos para que puedan aplicarlos en sus vidas. Estos conceptos suelen ser numerosos y muy variados: el desafío es no sobrecargar al alumno y a la vez provocar que se interese en el tema. Esto no siempre sucede, lo que genera profesionales escépticos a la aplicación de simulación para problemas reales, o que desarrollan expectativas irreales sobre qué es posible lograr mediante simulación, al punto que pueden incluso otorgarle capacidades excesivas en cuanto a sus posibilidades predictivas [Centeno01].

Si bien cada instructor diseña el curso de DES según los objetivos propuestos, la experiencia en el campo de la simulación, y su trasfondo personal, indican que hay elementos que deberían ser comunes a todos los cursos de DES, más allá de quién o dónde se dicte el curso. Asegurar que todos los elementos importantes sean incluidos en el curso, y que se consideren otros aspectos, como las habilidades de los estudiantes para resolver problemas y la o las herramientas que se utilizarán como soporte tecnológico para desarrollar el curso, debe ser el desafío a vencer. Así, los autores proponen un *framework* que se enfoca en la enseñanza de DES, y no en los algoritmos o en su aplicación. Este *framework* consta de una serie de elementos, cada uno de los cuales contiene actividades, métodos, herramientas de enseñanza y herramientas de aprendizaje [Garcia09]. Los elementos del *framework*, conocido como S.U.C.C.E.S.S.F.U.L, son los siguientes:

1. ***S**tandarize systems concepts knowledege*: es esencial que todos los alumnos entiendan lo mismo cuando se habla de sistemas (componentes, interacciones, entorno, etc.), y de la forma de analizar los problemas de un sistema.

2. ***U**nify simulation concepts*: es necesario impartir un conjunto unificado de conceptos sobre DES, el significado de eventos discretos, sus beneficios y limitaciones.

3. ***C**onvey relationships among system elements* (nivel de detalle): el diseño de un modelo debe enfocarse en el principal objetivo del proyecto de mejora. Debe establecerse un adecuado nivel de detalle en los modelos, que contenga los elementos mínimos para alcanzar los objetivos del proyecto.

4. ***C**omplement data collection and analysis concepts*: las herramientas de software permiten recolectar una gran cantidad de datos, muchas veces más de lo que se requiere, y muchas veces en un formato que hace muy difícil su aprovechamiento. Se deben incluir estrategias para recolectar sólo los datos requeridos, y para luego gestionarlos de manera eficiente.

5. ***E**stablish the importance of modeling accuracy*: el proceso de modelado debería permitir a los alumnos reducir la distancia entre la representación del sistema y el sistema real.

6. ***S**how modeling strategies*: el modelado de sistemas reales requiere el desarrollo de habilidades para lidiar con la complejidad a través de la abstracción y la síntesis del sistema real.

7. ***S**elect modeling software*: ¿cuánto tiempo se utilizará para aprender una herramienta de software específica? ¿cómo se integra esta herramienta con los conceptos teóricos y los objetivos del curso? ¿está disponible esta herramienta para los alumnos?

8. ***F**ormulate with post-processing analysis*: los alumnos deben aprender a analizar posibles escenarios, a elegir el mejor de ellos, a diseñar experimentos, etc. Los modelos deben plantearse desde el principio considerando estos aspectos.

9. ***U**ndertake model documentation activities*: para fomentar el reuso y la mejora, debe realizarse una apropiada documentación del modelo y del análisis del sistema (asunciones, simplificaciones, datos tomados, etc.)

10. ***L**ead students through a comprehensive case study*: análisis de un caso real, desarrollo de un proyecto de simulación de principio a fin.

A partir de los elementos incluidos en el *framework*, se destacan algunas observaciones interesantes:

a) existe una gran variedad temática en el curso de DES, que va desde conceptos de estadística hasta modelado y análisis de datos. Desde luego, todos estos conceptos deben dictarse en un tiempo bastante acotado;

b) el docente debe considerar la disparidad de conocimientos y experiencia de los alumnos. Se organizará el curso a partir de esta disparidad y se buscará homogeneizar los conceptos centrales antes de avanzar con los temas propios del curso de DES;

c) la herramienta de software utilizada para simulación debe servir como apoyo al curso, pero no es el objetivo del curso aprender a utilizar esta herramienta. Si esta herramienta permite relacionar de manera natural los temas teóricos con conceptos prácticos, mucho mejor;

d) se espera que los alumnos adquieran muchas habilidades prácticas a partir de los conceptos teóricos y el uso de las herramientas de aprendizaje. Si bien el aprendizaje se hace por etapas, es importante contar con un trabajo integrador al finalizar el curso, o con diversos trabajos que permitan integrar los conceptos aprendidos en varias etapas.

Estas observaciones fueron tenidas en cuenta a la hora de desarrollar y diseñar la herramienta de software y el curso de Modelos y Simulación en el contexto de esta tesis.

Ingolf Ståhl [Ståhl00], en su trabajo "¿Cómo debemos enseñar simulación?", realiza un detallado análisis sobre los tipos de estudiantes que aprenden simulación, las razones para incluir estos contenidos en las distintas universidades, y compara simuladores animados con lenguajes de simulación. Allí, Ståhl destaca que la simulación es una herramienta cada vez más necesaria en una gran variedad de ámbitos, pero que su enseñanza conlleva un elevado costo de aprendizaje, y que en muchos casos los estudiantes no encuentran gran utilidad o posibilidades de aplicación en el futuro, lo que hace que no consideren necesaria la inversión de tiempo y esfuerzo, y, por ende, decidan no aprender simulación. Por este motivo, el autor considera necesario reducir el costo, en particular en términos de tiempo, para aprender simulación.

Entre los tipos de estudiantes, Ståhl identifica tres grandes grupos. En el primer grupo se encuentran los estudiantes del área de sistemas e informática, que conocen bien

lenguajes de programación y conceptos de estadísticas. El segundo grupo abarca a los estudiantes de ingeniería, que pueden contar con conocimientos básicos de programación y sólidos conocimientos de estadísticas. Por último, en el tercer grupo se encuentran los estudiantes de negocios, que por lo general no poseen conocimientos sobre programación y en muchos casos tampoco poseen una sólida base estadística. El desafío para el primer grupo, y en parte para el segundo grupo, es adoptar un nuevo paradigma de programación, dado que los lenguajes de simulación son por lo general muy diferentes a los lenguajes de programación tradicionales. Sin embargo, estos alumnos conocen sobre proyectos de software, etapas de análisis, diseño, modelado e implementación, programación y complicación, pruebas, reuso, mantenimiento, etc. En el tercer grupo, el esfuerzo para elaborar un proyecto de simulación es mucho mayor, así como también el de demostrar la importancia del aprendizaje de esta herramienta para el estudio de sistemas reales. Esto se debe a que, por lo general, los alumnos logran apreciar la verdadera potencia de esta herramienta una vez que han desarrollado un proyecto completo, desde la recolección inicial de datos hasta la etapa de experimentación y análisis de resultados.

Si se toma en consideración la clasificación de los tipos de estudiantes, es evidente que el tipo de software a aprender diferirá en cada caso. Los estudiantes de informática y sistemas, con conocimientos de C++ o Java, procederán rápidamente a aprender los detalles de un sistema de simulación, y pueden utilizar estos lenguajes de propósito general (General Purpose Language, GPL) directamente para sus modelos de simulación. Sin embargo, el autor afirma que posee muy poca experiencia con este grupo de estudiantes. Para los estudiantes de otras ingenierías, los simuladores animados o interactivos son comúnmente usados. En estos simuladores, el modelo se genera mediante la selección de bloques de construcción, que consisten en íconos que representan entidades como máquinas y servicios. Al hacer clic sobre un ícono, estos simuladores presentan un menú o diálogo que provee algo de información sobre las características específicas del proceso de esta máquina. Esto permite reducir considerablemente el tiempo de aprendizaje, pero presenta una limitación importante: el esfuerzo de aprendizaje y modelado se incrementa rápidamente cuando se generalizan las ideas, a fin de trabajar con modelos más allá de los que el software de simulación utilizado considera. Finalmente, para los alumnos del área de negocios, se

recomienda un paquete de simulación en la forma de un lenguaje de simulación, como GPSS, SIMAN [Pegden95] y SLAM [Pegden79][PritsKer99]. Si bien los lenguajes de simulación requieren una etapa de aprendizaje, esta es mucho menor comparada con un GPL.

En la última parte de su trabajo, Ståhl lista una serie de criterios para software de simulación utilizado en educación. Estos criterios se agrupan en 8 categorías generales. A continuación, se listan estas categorías y los principales requerimientos incluidos en cada una:

1. *Facilidad de aprendizaje*: los estudiantes deben concentrarse en el modelado, y no en cuestiones de sintaxis; se debe promover el aprendizaje por intuición y alentarlos a investigar la herramienta en vez de poner barreras, se debe permitir el uso de ejemplos y el estudio de casos concretos; se debe ayudar al alumno a interpretar el comportamiento de la simulación o el sistema de encolamiento, entre otras características.

2. *Facilidad de ingreso de datos*: es preferible el uso de interfaces gráficas, del ratón para arrastrar y soltar, se debe ayudar a identificar los símbolos y elementos que pueden utilizar los alumnos y asociarlos con conceptos vistos en clase. Se recomienda el uso de diálogos para la carga de datos, y que estos diálogos incluyan información sobre los operandos de los bloques, entre otras posibilidades.

3. *Facilidad para interpretar la salida*: la salida debe estar adaptada para estudiantes, y no para usuarios profesionales; se valora el uso de gráficos e histogramas, así como también el uso de animaciones y funciones específicas para comprender cómo el programa de simulación funciona internamente.

4. *Facilidad para realizar replicaciones y experimentos*.

5. *Programación segura*: la interfaz de trabajo debe asistir al usuario para evitar errores de compilación, y también para evitar errores durante la ejecución. Es importante contar con herramientas de *debug* y de seguimiento detallado de la ejecución del programa.

6. *Eficiencia*: si bien en este contexto no es tan importante, es preferible que los tiempos de ejecución no sean excesivamente largos para no desalentar a los estudiantes.

7. *Disponibilidad*: el programa debe estar disponible para varias plataformas. También es deseable contar con paquetes de bajo costo, que puedan ser adquiridos por estudiantes, o incluso versiones gratuitas para ámbitos académicos. Se recomienda especialmente que esté disponible en la web, principalmente por tres razones: 1) los estudiantes siempre accederán a la versión más actualizada; 2) podrán utilizar el sistema incluso fuera de la universidad; 3) pueden visualizar el programa antes de descargarlo e instalarlo en sus computadoras.

8. *Potencial avance hacia otras herramientas*: los estudiantes deben poder migrar hacia herramientas profesionales sin mayores inconvenientes.

Finalmente, el autor recomienda WebGPSS para entornos web, así como también WinGPSS (MS Windows), LinGPSS (Linux) y MacGPSS (Mac). Todas esas herramientas fueron desarrolladas por el mismo autor y se basan en micro-GPSS (también desarrollado por él): un motor de simulación basado en GPSS para entornos de aprendizaje, y cumplen con la mayoría de los criterios definidos. Estas herramientas son analizadas con mayor detalle más adelante en esta sección.

En la Facultad de Ciencias Organizacionales de la Universidad de Belgrado (República de Serbia), los profesores Despotović, Radenković y Barać propusieron un implementación de GPSS para entornos de *e-learning*, a la que denominaron FONWebGPSS [Despotović09]. El curso de Simulación y Lenguajes de Simulación se dicta para alumnos de Sistemas de Información y Tecnologías de Internet, y se realiza en parte de manera presencial, momento en el que se discuten casos de estudio y problemas relacionados con el área de simulación continua y de eventos discretos, y en parte a distancia, a través del sistema de gestión de enseñanza Moodle[3]. Se utiliza la implementación GPSS/FON[Zikic92], que no tiene funciones para integrarse a entornos de enseñanza en línea, para proveer comunicación entre estudiantes y docentes, o para permitir que los docentes asistan a los alumnos mientras realizan los trabajos.

[3] Moodle: open-source community-based tools for learning. Página web: https://moodle.org/.

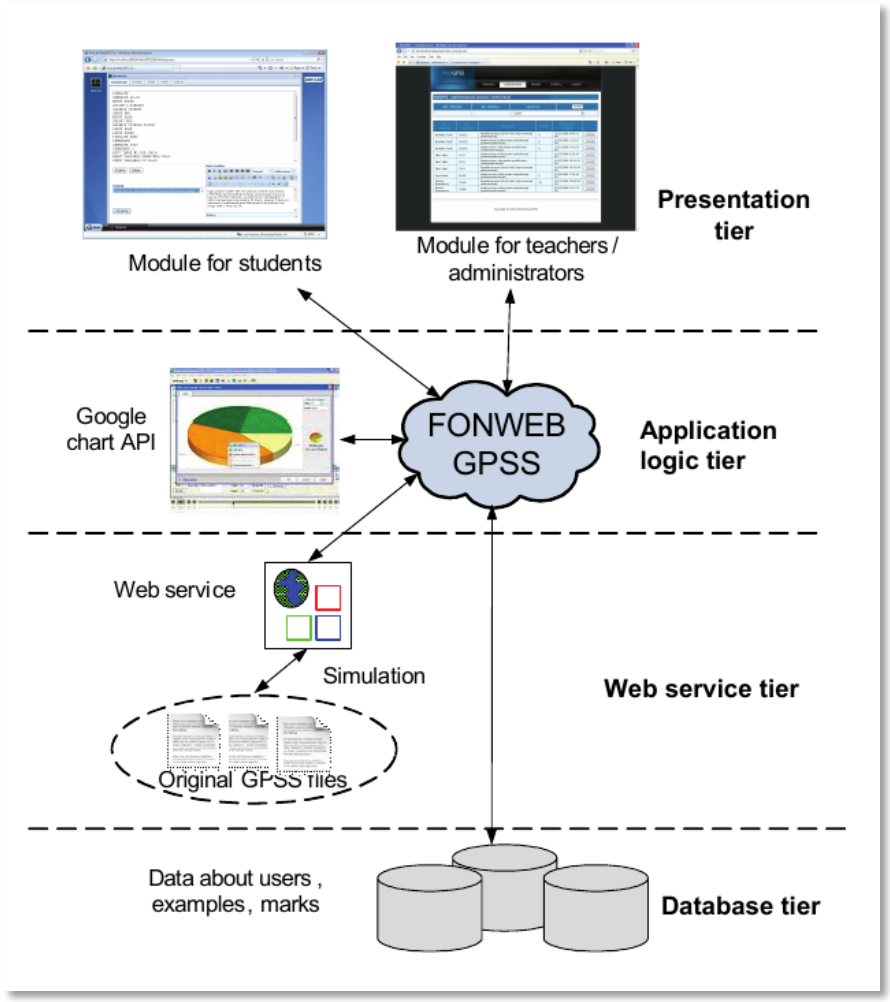


Imagen 1: Arquitectura de FONWebGPSS [Despotović09]

FONWebGPSS, desarrollada en el laboratorio para simulación, permite a los estudiantes configurar modelos de simulación, ejecutarlos y analizar los resultados a través de una interfaz web. También les permite enviar soluciones para casos y ejercicios de simulación de eventos discretos. Por otro lado, esta aplicación permite al docente monitorear y asistir a los estudiantes en sus actividades. La interfaz de trabajo para docentes les permite también crear, probar y gestionar casos y problemas relacionados con simulación de eventos discretos, crear reportes y realizar algunas actividades básicas de administración, como la gestión de cuentas de usuario.

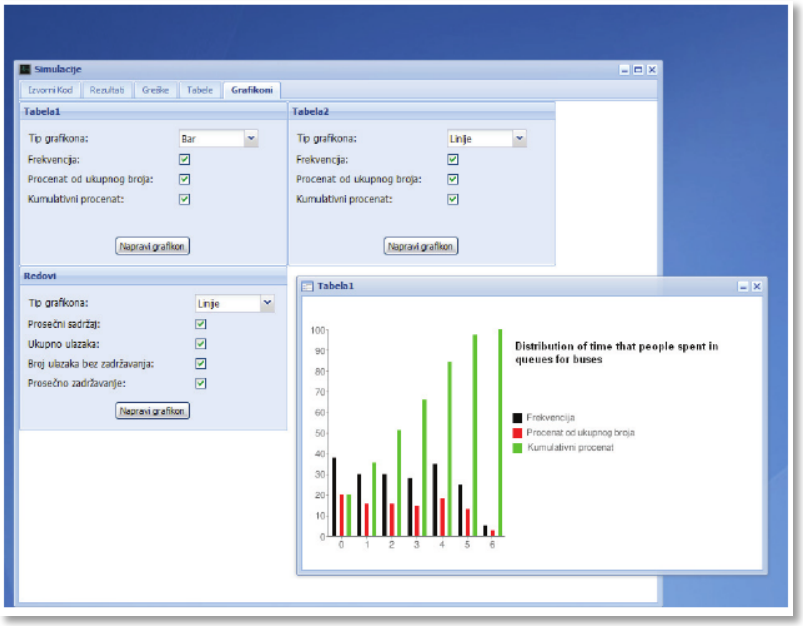


Imagen 2: Visualización de resultados de FONWebGPSS [Despotović09]

Esta aplicación cuenta con una capa intermedia de presentación, que utiliza tecnologías web como HTML/XML y Javascript (mediante el *framework* ExtJS[4]), y que se encarga de generar la interfaz dinámica de usuario (Imagen 1). Existe asimismo una capa de lógica de aplicación que se encarga de recibir y manejar los *requests* desde los clientes web, y comunicarse con la base de datos, invocar al motor de simulación, recibir los resultados de la simulación y enviarlos al componente que genera gráficos e histogramas a partir de estos datos (Imagen 2). Para la ejecución de la simulación, se envía el código GPSS hacia la aplicación GPSS/FON a través de un *web service* que invoca a un método específico de esta aplicación. Este *web service* utiliza el estándar XML para la comunicación entre las partes.

[4] Sencha Ext JS JavaScript Framework for Rich Desktop Apps. Página web: http://www.sencha.com/products/extjs.

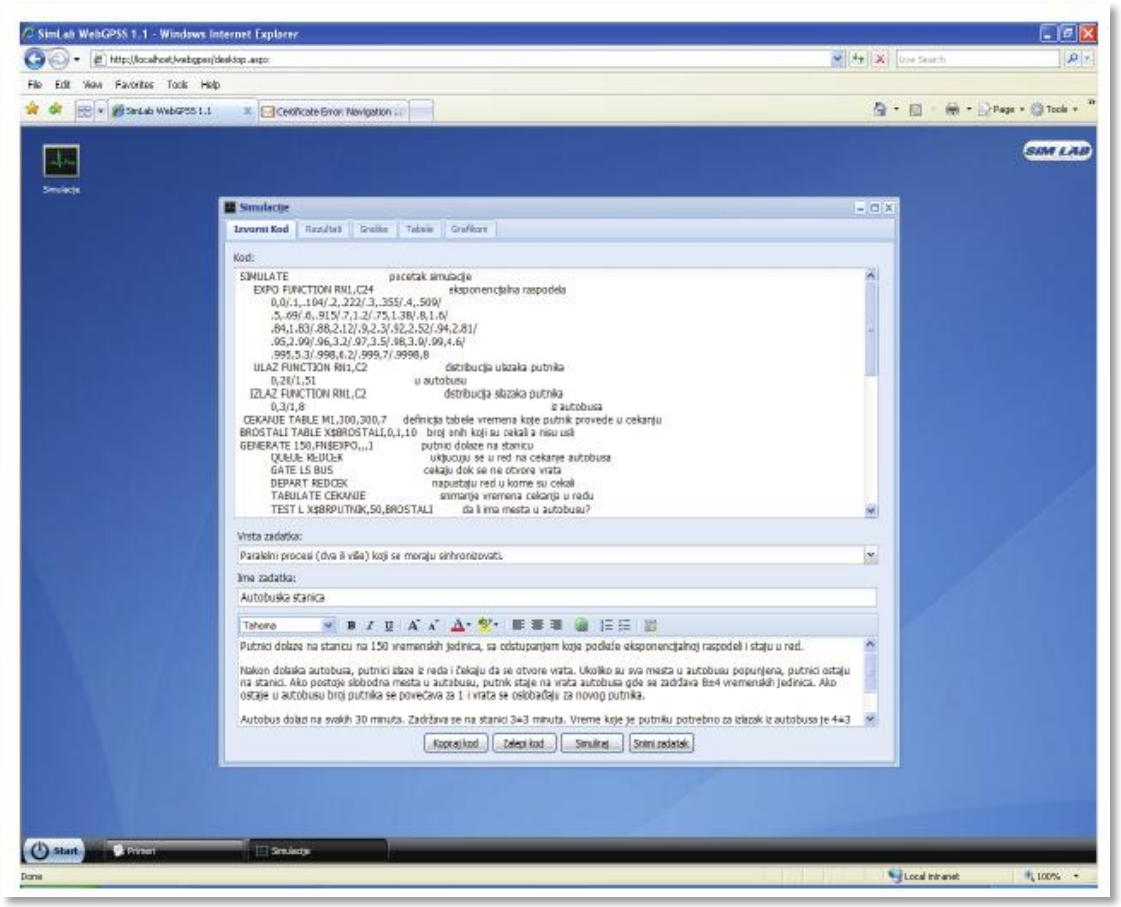


Imagen 3: Editor de código de FONWebGPSS [Despotović09]

Una característica interesante de FONWebGPSS es que se integra al Moodle de la universidad, lo que permite a los docentes integrar los contenidos teóricos, casos de estudio y ejercicios propuestos con el entorno para crear, ejecutar y analizar los resultados. Además, los autores de este trabajo destacan la importancia de utilizar entornos web para la enseñanza en esta área, lo cual evita que los alumnos deban instalar y actualizar el software, y permite a los docentes controlar más de cerca el proceso de enseñanza. Por otro lado, es importante destacar que los autores han puesto énfasis en la integración del motor de simulación con su entorno web de aprendizaje y en la visualización de resultados. FONWebGPSS no implementa una interfaz de trabajo específica para GPSS, ni aporta asistencia alguna a los estudiantes durante la creación de modelos GPSS. En cambio, presenta un área de texto plano (Imagen 3), en la cual los alumnos escriben el código GPSS, que es enviado al motor de simulación residente en el servidor, para luego recibir los resultados de dicha simulación.

A. M. Zikic y B. Lj. Radenkovic [Zikic96] presentaron, en el año 1996, un lenguaje de programación diseñado específicamente para la enseñanza en el área de simulación de sistemas de eventos discretos. Este lenguaje, llamado Sistema Interactivo para Simulación de Eventos Discretos, o ISDS, se basa en los mismos conceptos de GPSS pero utiliza una interfaz de usuario interactiva y una sintaxis similar a Pascal. Luego de muchos años de enseñar en esta área, los autores realizaron un seguimiento del progreso de los estudiantes en el aprendizaje de conceptos de GPSS, así como también el tiempo promedio necesario para adoptar una aproximación completamente nueva a la programación. Este seguimiento les permitió realizar observaciones muy interesantes:

1. Los estudiantes aprenden programación a partir de lenguajes como Pascal o C, que emplean sistemas de tipos de variables y objetos muy estrictos y explícitos.

2. Por otro lado, GPSS no posee ni declaración explícita ni tipado estricto de variables, sino que utiliza una combinación de ambos, entre implícito y explícito.

3. Una aproximación estructurada a la programación, desarrollada utilizando Pascal como lenguaje principal de programación, se convierte en algo natural para los estudiantes, mucho antes de intentar enseñarles simulación de eventos discretos.

4. La sintaxis de GPSS es del estilo del lenguaje Assembler [Stabley90], con muchas etiquetas y sentencias de transferencia. Al momento de su creación, la programación estructurada no existía.

Claramente, las conclusiones 1 y 3 están en conflicto con las conclusiones 2 y 4 [Zikic96]. Bajo estas condiciones, los autores aseguran que el origen del problema está en la sintaxis y la estructura de GPSS, en particular cuando los estudiantes ya tenían conocimientos previos sobre técnicas y conceptos modernos de programación. Como solución a este problema, proponen un nuevo lenguaje de programación que satisface los siguientes cuatro requerimientos:

1. Económico y costeable para que un alumno promedio pueda practicar en su casa.

2. La sintaxis del lenguaje debe poseer una especificación estrictamente dividida entre el modelo de base de datos, la estructura del modelo y el control del proceso de simulación.

3. El entorno de trabajo debe ser interactivo, con un editor, procesador y analizador

de resultados.

4. Modelo de *debug* rápido y simple.

La solución propuesta se denominó ISDS (Interactive System for Discrete-event System), y toma algunos aspectos de Passim [Ueno80][Barnett82], una librería que permite incorporar sentencias de GPSS en programas escritos en Pascal. ISDS mantiene algunos nombres de instrucciones de GPSS, mientras que en los casos donde las instrucciones son nuevas, en ISDS se debe o bien a que no existen en GPSS, o bien a que causan confusión en los estudiantes. ISDS posee una sintaxis formal estricta, que sirve para verificar si los tipos de los operandos se corresponden con la definición de las funciones. También posee un conjunto de instrucciones IF-THEN-ELSE-CASE, que permite escribir lógicas complejas evitando los saltos de GPSS a través de GATE, TEST, TRANSFER, entre otros bloques que provocan saltos bajo determinadas condiciones. Además de estos condicionales, ISDS también posee bloques al estilo GPSS.

Uno de los cambios más radicales de ISDS es que las entidades estáticas (Facility, Savevalue, Logic switch, etc.) deben ser explícitamente definidas en un segmento separado del programa, lo que reduce los errores por el uso de entidades inexistentes. Asimismo, al igual que GPSS, los programas escritos en ISDS poseen funciones para recopilar estadísticas de manera automática, relativas a todas las entidades permanentes definidas explícitamente por el usuario.

En comparación con GPSS/FON y con Passim, ISDS mostró una clara mejora de performance (medida en horas requeridas para completar los ejercicios) en relación al segundo, del orden del 300% al 400%, puesto que los estudiantes mostraron dificultades tanto al comprender los conceptos de GPSS como al implementar sus modelos. Al compararse con GPSS/FON, no se detectó una gran mejoría respecto a la facilidad de aprendizaje, pero sí se observó una mejoría cercana al 50% del tiempo requerido para implementar los modelos. Sin embargo, una de las principales desventajas de esta aproximación es que, una vez completado el curso, los alumnos debían aún aprender GPSS si querían utilizarlo en el ámbito profesional. Además, ISDS no se distribuye de manera libre, y solo se entrega a docentes bajo pedido explícito para ser utilizado en investigación y dentro de sus cursos de simulación.

Fonseca i Casas y Casanovas [Fonseca09] proponen en su trabajo una aplicación de

escritorio para la enseñanza de GPSS a la que llamaron JGPSS. Los autores destacan la importancia, desde el punto de vista del aprendizaje, del conocimiento en detalle del motor de simulación, e incluso proponen que los alumnos lo modifiquen o generen el suyo propio: "todas las versiones de GPSS permiten que los estudiantes comprendan cómo funciona el sistema GPSS y son excelentes alternativas para introducir a los estudiantes al mundo de la simulación discreta [...]. Sin embargo, para comprender en profundidad cómo funciona el motor de simulación, conocimiento necesario para un estudiante de ciencias de la computación, el mejor ejercicio es construirlo por ellos mismos". JGPSS permite a los alumnos acceder a la implementación del motor de simulación, e insertar su propio código Java para alterar esta implementación.

JGPSS se caracteriza por presentar, en su interfaz gráfica de usuario (Imagen 4), algunos bloques de GPSS, en particular los más utilizados, como GENERATE/TERMINATE, QUEUE/DEPART, y SEIZE/RELEASE. Estos bloques son presentados a partir de figuras, que muestran el nombre del bloque y los parámetros que los mismos requieren.

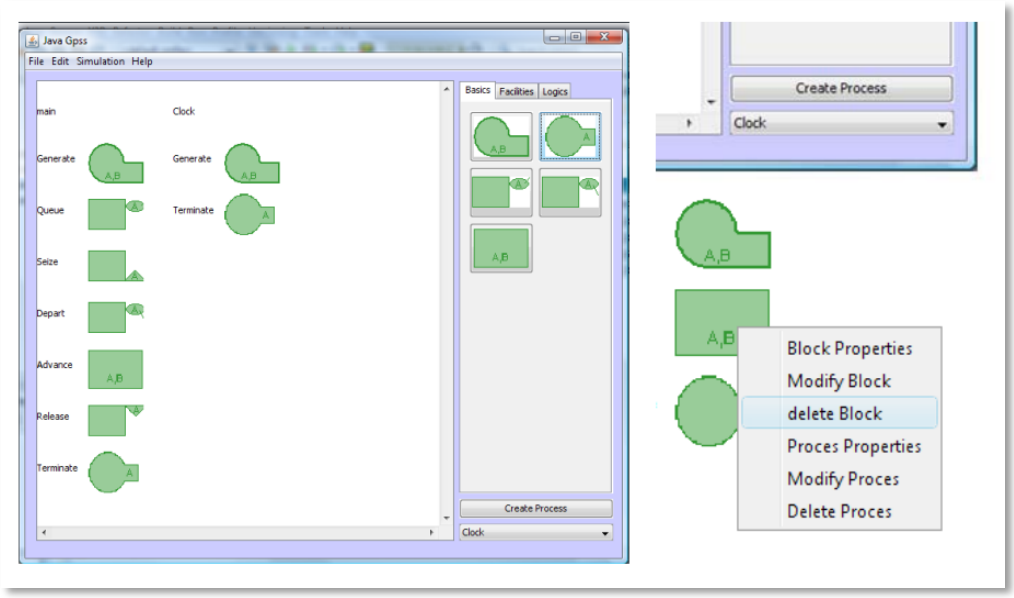


Imagen 4: Interfaz de usuario para la construcción de modelos de JGPSS [Fonseca09]

Para simplificar al alumno la tarea de asociar los bloques con las entidades, JGPSS organiza los bloques disponibles en diferentes solapas, una para bloques básicos, otra

para facilidades y otra para bloques lógicos. Esta idea también fue aplicada en el desarrollo que se presenta en esta tesis, aunque ampliada a una mayor cantidad de bloques y a categorías más específicas.

Al seleccionar un bloque, JGPSS presenta un diálogo (Imagen 5) que permite al usuario cargar uno a uno los parámetros del bloque en cuestión. Si bien esto simplifica la carga de los parámetros, no se brindan ayudas contextuales que permitan a los alumnos conocer el uso que se le dará a cada parámetro, ni tampoco información que indique qué hace o para qué sirve el bloque seleccionado. Los bloques, una vez insertados, componen un gráfico que puede ser fácilmente reorganizado arrastrando y soltando los mismos en los distintos lugares del diagrama.

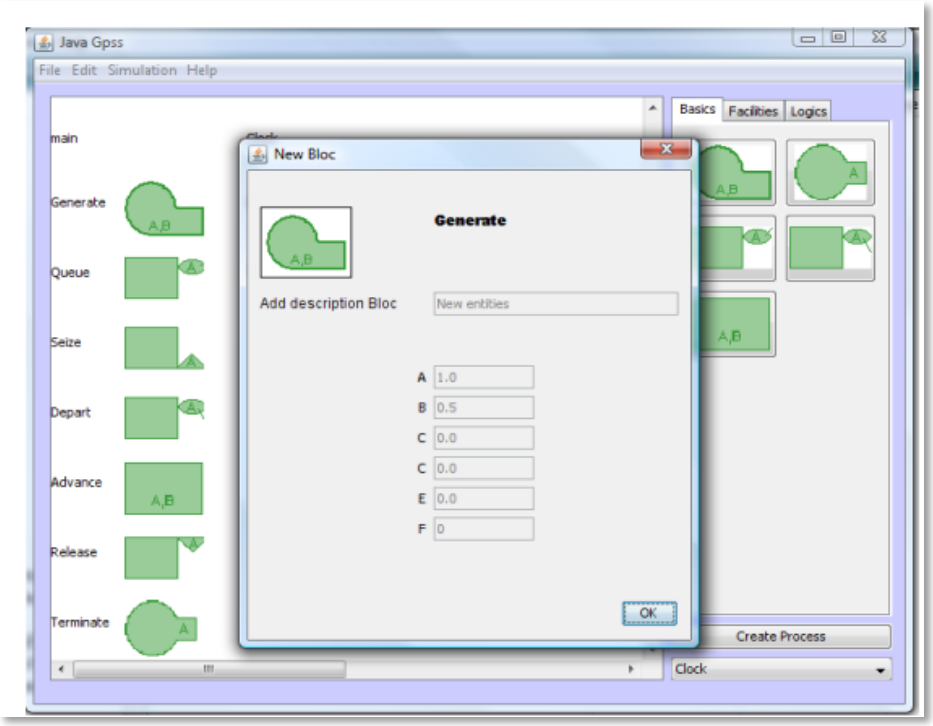


Imagen 5: Uso de diálogos para insertar bloques en JGPSS [Fonseca09]

El objetivo principal de JGPSS es que los alumnos enfoquen su atención para programar el motor de simulación; la aplicación presenta algunas áreas para insertar código Java y cuenta con una guía de codificación para los estudiantes. Utilizando cuestionarios SEEQ (Student Experience of Education Questionnaire, Cuestionarios de Experiencia de Educación de Estudiantes), los autores detectaron desde el comienzo que

los alumnos observaban resultados, lo que los mantenía más motivados [Fonseca09].

Henry Herper e Ingolf Ståhl presentaron en el año 1999 [Herper99] dos herramientas para la enseñanza de GPSS, construidas sobre Micro-GPSS, una versión de GPSS que posee algunas características que la hacen adecuada para su uso en el ámbito académico. En el trabajo, se presentan las ventajas del uso de dicha herramienta en el curso, que se detallan a continuación; sin embargo, en el mismo no se mencionan estrategias o métodos que acompañen el uso de estas herramientas para aprovechar al máximo todas sus características pedagógicas en el marco de un curso de simulación.

Micro-GPSS es el resultado de un proceso de desarrollo educacional basado en 20 años de enseñanza de GPSS a más de 5000 estudiantes. Esta aplicación, que se ejecuta en MS-DOS [Cooper02], fue diseñada para asistir en la enseñanza en el área de simulación y, por lo tanto, posee algunas características que simplifican la enseñanza de GPSS. Entre estas características, se destaca la considerable reducción en la cantidad de bloques que posee esta implementación. Se han eliminado bloques con el objetivo de reducir la redundancia entre bloques similares. Por ejemplo, utiliza un bloque LET que cubre ASSIGN, SAVEVALUE, INITIAL, BLET y LET de GPSS/H. Además, en algunos casos, alteraron la sintaxis original de GPSS buscando cierta similitud con estructuras ya conocidas por los alumnos. En este sentido, se reemplazaron algunos bloques como GATE y TEST por sentencias más parecidas a lenguajes procedurales; por ejemplo, el típico TEST de GPSS

TEST GE C1,100,*NEXT*

se transforma en una sentencia condicional *if-then* en Micro-GPSS:

IF C1<100,*NEXT*

Esta implementación de GPSS no posee una interfaz gráfica de usuario (Graphic User Interface, GUI). Los modelos se desarrollan en editores de texto externos y se almacenan en archivos de texto plano, los cuales son luego pasados al motor de simulación por medio de la línea de comandos. Sin embargo, en 1998 comenzó el desarrollo de algunas interfaces gráficas que interactúan con este simulador. Dos implementaciones fueron completadas, una de ellas basada en la web, WebGPSS, y la otra como una aplicación de escritorio, WinGPSS, compatible con la familia de sistemas operativos MS Windows.

Ambos desarrollos seguían una idea común: los modelos de desarrollaban de manera interactiva, mediante gráficos que representaban los bloques y diálogos que permitían cargar los parámetros de dichos bloques (Imagen 6). Los modelos generados eran luego enviados hacia el motor de simulación GPSS.EXE, que debió ser modificado para permitir su ejecución tanto en la web como en entornos Windows. Además, se incluyeron otras mejoras en el simulador original, como la entrada de datos interactiva, interrupción de programas que se ejecutaban por mucho tiempo, re-ejecución simple de programas, definición de funciones simples (Imagen 7), generación de archivos con datos para la construcción de gráficos e histogramas, y la generación de datos para construir una animación que muestre el movimiento de las transacciones a través del modelo.

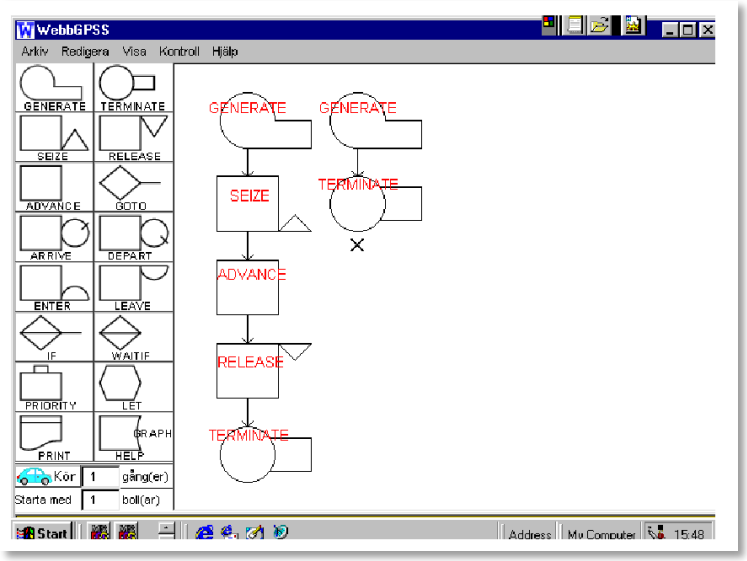


Imagen 6: Interfaz gráfica de usuario de WebGPSS [Herper99]

WebGPSS fue pensado para que los alumnos puedan desarrollar pequeños proyectos de simulación. Este programa representa un esfuerzo conjunto del Departamento de Economía Empresarial de la Escuela de Economía de Estocolmo (Stockholm Economics School, SSE), Flux Software Engineering, Ronneby Soft Center, la Universidad de Karlskrona-Ronneby (HKR) y la ciudad de Ronney. La versión básica, completada para el año 1999, poseía tres componentes principales:

1. Una interfaz de usuario en forma de un *applet* basado en Java, el cual se ejecuta dentro de los navegadores web en las computadoras de los estudiantes y permite

construir los modelos GPSS de manera interactiva. Posee algunos textos de ayuda integrados, y ofrece enlaces de acceso a tutoriales de GPSS.

2. Los ajustes al micro-GPSS original, para hacerlo compatible con la ejecución a través de la web.

3. Un sistema que hace posible la transferencia de programas construidos en el *applet* en la computadora del estudiante hacia un servidor, luego ejecutar el programa utilizando GPSS.EXE en dicho servidor, y finalmente transmitir los resultados de la simulación hacia la computadora del cliente. Esta interacción entre el cliente y el servidor se realizaba mediante el envío de archivos, que contenían tanto el modelo como los resultados.

Es preciso observar aquí que, para el año 1999, muchas de las tecnologías actuales tan difundidas en la web como Javascript y AJAX, eran muy rudimentarias o no existían, lo cual explica por qué este proyecto requirió el esfuerzo de tantas instituciones y organizaciones.

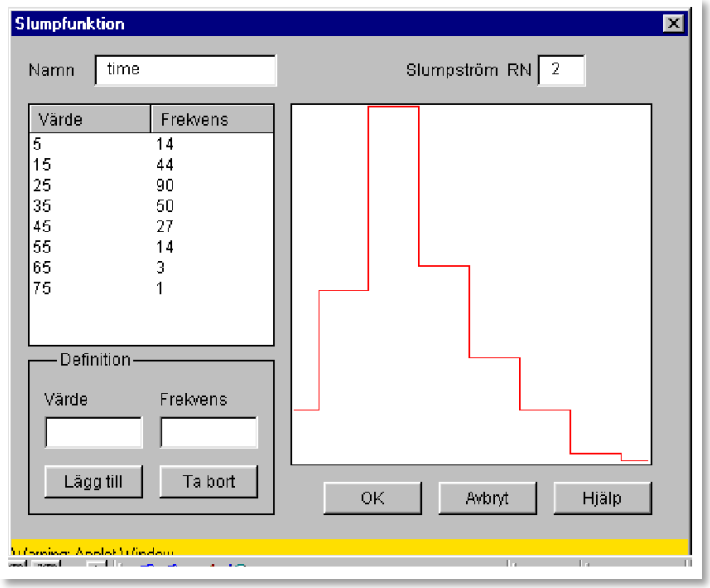


Imagen 7: Definición de funciones aleatorias en WebGPSS [Herper99]

Entre las principales ventajas de esta solución, se destaca que los alumnos siempre utilizarán la última versión tanto de la interfaz de usuario como del motor de simulación, dado que no deben instalar el software en sus computadoras, que los estudiantes

pueden a acceder a este entorno de simulación desde cualquier lugar conectado a Internet, que el uso de Internet favorece el aprovechamiento de otras actividades web como el chat entre usuarios del software, y principalmente el uso del lenguaje Java, lo cual brinda la posibilidad de generar un software independiente de la plataforma subyacente [Herper99].

WinGPSS consiste en una interfaz de usuario gráfica, diseñada exclusivamente para entornos Windows (Imagen 8). Este proyecto, desarrollado en Delphi, fue liderado por Henry Herper y contó con la ayuda de A. Krüger y H. Schielfke, del Instituto para Simulación y Gráficos, de la Universidad Otto-von-Guericke, Magdeburgo (Alemania). Para este desarrollo se tomaron en consideración algunas desventajas que presentaba WebGPSS en comparación con un sistema de escritorio. Estas incluían: 1) el alto costo de conectar a los alumnos de las universidades a servidores remotos en muchos países europeos; 2) el excesivo tiempo requerido para descargar el *applet* en el navegador; y 3) la imposibilidad de ejecutar esta aplicación sin conexión a Internet. En la actualidad, es evidente que los problemas 1 y 2 ya no existen, y que el problema 3 tiende a desaparecer con la posibilidad de acceder a Internet desde casi cualquier lugar mediante tecnologías como 3G [Dahlman08] y WiMAX [Vaughan04]. Sin embargo, en 1999, estos problemas originaron una serie de obstáculos que fomentaron el desarrollo de WinGPSS. Existe también un problema adicional de WebGPSS, relacionado con la tecnología utilizada para su desarrollo. Al ser un *applet* Java, no es posible almacenar archivos en el disco rígido del cliente debido a su política de seguridad [Jamsa02]. Y, por razones de costo, tampoco era viable en ese momento hacerlo en el servidor. Esto requería que los alumnos copien el programa generado en la ventana de WebGPSS y lo inserten en un editor de textos (como Notepad o similares), y de allí guardarlo en sus computadoras o en un dispositivo externo.

Por último, una de las ventajas principales de la solución de escritorio radica en la posibilidad de interactuar con otras aplicaciones instaladas en el equipo, en particular con MS Excel. WinGPSS puede enviar grandes conjuntos de datos hacia Excel por medio de una interfaz para programadores, y desde allí los alumnos pueden realizar análisis más complejos a partir de los resultados de la simulación.

Implementaciones de GPSS

Además de los desarrollos previamente mencionados, existen algunas implementaciones profesionales de GPSS. La mayoría de las aplicaciones han sido completadas hace varios años, y no se han incorporado nuevas funciones o mejoras desde entonces. La única excepción aquí es GPSS World, cuya empresa desarrolladora, Minuteman Software, ha continuado el desarrollo y lanzado nuevas versiones hasta el año 2010 (versión 5.2.0).

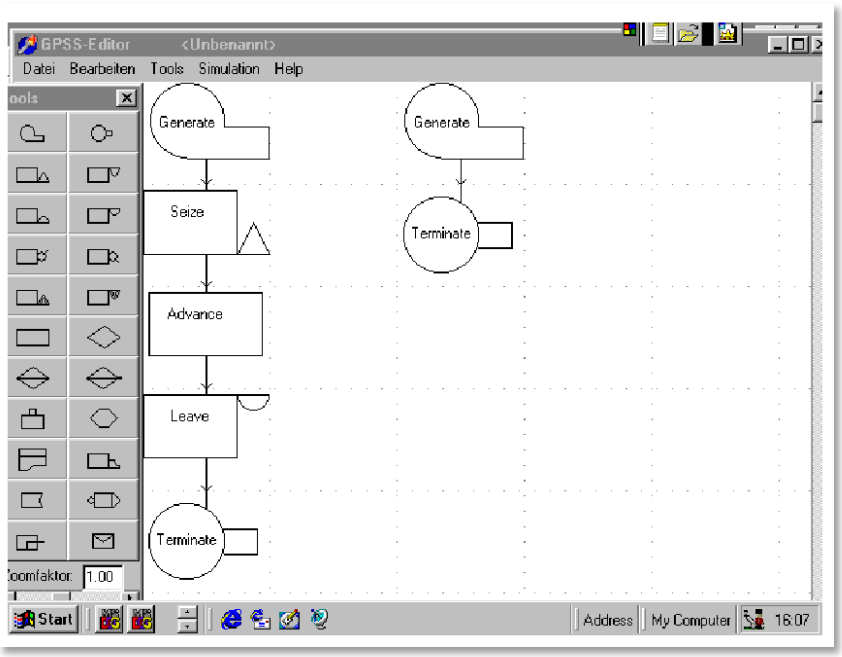


Imagen 8: Interfaz gráfica de usuario de WinGSS [Herper99]

Las distintas implementaciones de GPSS incluyen herramientas de análisis avanzadas, y poseen funciones y tecnologías que les permiten abarcar problemas más complejos o desarrollar soluciones que hacen mejor uso de las tecnologías actuales: lenguaje de programación procedural embebido, librería de funciones, ejecución multi-hilo (*multithreaded*), memoria virtual, aplicaciones multiplataforma y simulación distribuida son algunos de los principales avances introducidos por las distintas versiones más actuales de GPSS. A continuación se realiza una breve reseña de las principales implementaciones de GPSS.

GPSS/H

GPSS/H es un desarrollo de Wolverine Software[5], y corre en cualquier versión de MS Windows incluyendo Windows XP, Vista, y Windows 7. Este desarrollo cuenta con una versión completa, cuyo precio oscila entre los 300 y los 4000 dólares según la licencia, y una versión gratuita para estudiantes, que les permite realizar modelos de tamaño reducido: hasta 125 bloques y 250 sentencias en total (bloques más comandos), 300 transacciones simultáneas como máximo, y hasta 32KB de memoria compartida.

```
gonetil@gonetil-laptop:~/.wine/drive_c/Wolverine/GPSSH$ cat MODEL12B.lis
    Student GPSS/H Release 3.70 (PR042)      2 May 2013   12:58:56     File: MODEL12B.GPS

   Line# Stmt#  If Do  Block#  *Loc    Operation      A,B,C,D,E,F,G   Comments

       1     1                        SIMULATE                                    Case Study 12B
       2     2                 *                                  An Equipment Balancing Problem
       3     3                 *                          (3 Cranes Serving a Group of 15 Machines;
       4     4                 *                        Finishing-Machine Utilization Not Measured)
       5     5                 *                                        Base Time Unit: 1 Minute
       6     6                 ************************************************************************
       7     7                 *         Control Statements (STORAGE)                                 *
       8     8                 ************************************************************************
       9     9                 *
      10    10                  CRANES   STORAGE    3          provide 3 cranes in the system
      11    11                *
      12    12                ************************************************************************
      13    13                *         Model Segment 1 (The Work Cycle for Operators)                *
      14    14                ************************************************************************
      15    15                *
      16    16             1            GENERATE   0,,,15,10   provide 1 operator per
      17    17                *                                finishing machine (Priority = 10)
      18    18             2   NEXTCYCL ADVANCE    60,10       Process 1 time
      19    19                *
      20    20             3            ENTER      CRANES      request/capture a crane
      21    21             4            ADVANCE    15,5        intermediate handling time
      22    22             5            LEAVE      CRANES      let this crane go
      23    23                *
      24    24             6            PRIORITY   5           drop Priority for next crane use
      25    25                *
      26    26             7            ADVANCE    90,20       Process 2 time
      27    27                *
      28    28             8            ENTER      CRANES      request/capture a crane
      29    29             9            ADVANCE    30,5        unload; store; fetch; reload time
      30    30            10            LEAVE      CRANES      let this crane go
      31    31                *
      32    32            11            PRIORITY   10          raise Priority for next crane use
      33    33                *
      34    34            12            TRANSFER   ,NEXTCYCL   go do Process 1 on next casting
      35    35                *
      36    36                ************************************************************************
      37    37                *         Model Segment 2 (Run-Control Xact)                           *
      38    38                ************************************************************************
      39    39                *
      40    40            13            GENERATE   2400        control Xact comes after 400 hours
      41    41            14            TERMINATE  1           reduce the TC value by 1,
      42    42                *                                ending Xact movement
      43    43                *
      44    44                ************************************************************************
      45    45                *         Run-Control Statements                                       *
      46    46                ************************************************************************
      47    47                *
      48    48                          START      1            set TC = 1;  start Xact movement
      49    49                *
      50    50                          END                     end of Model-File execution
```

Imagen 9: Visualización de código en GPSS/H

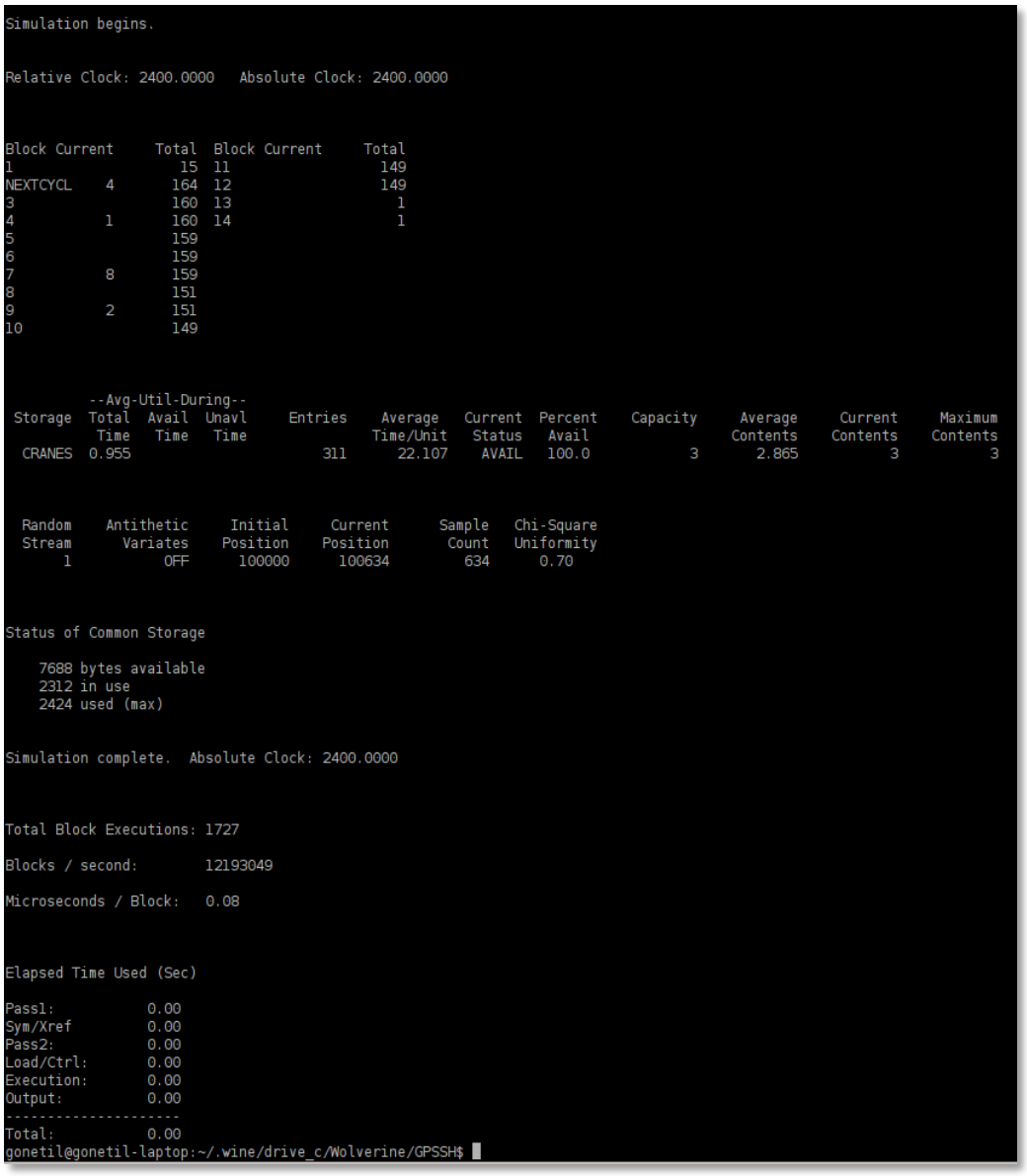
[5] Página web: http://www.wolverinesoftware.com/GPSSHProducts.htm.

GPSS/H fue desarrollado en 1977 para ser ejecutado en *mainframes* de IBM, luego actualizado para computadoras VAX en 1983, equipos de trabajo soportados por Unix (1986), y finalmente la PC de IBM. La versión personal de GPSS fue lanzada en 1988, la versión para estudiantes en 1989, y la profesional en 1990. GPSS/H no posee un entorno de desarrollo integrado disponible para el usuario. Los modelos son desarrollados con un editor de texto externo, y almacenados en archivos de texto plano. Mediante la línea de comandos, el programador invoca al compilador de GPSS/H para compilar el código fuente directamente sobre la memoria del sistema e inmediatamente ejecutarlo [Crain99]. La ejecución de la simulación se realiza de fondo, o sea que no se muestra ningún *feedback* al programador sobre la ejecución. Una vez finalizada, se generará un archivo con el mismo nombre del archivo de entrada y extensión .lis, (Imagen 9, Imagen 10) que contiene los resultados de la ejecución. Los desarrolladores de esta herramienta sugieren que, mientras que la construcción de modelos visuales, por medio de interfaces icónicas y diálogos, permite incluso a novatos en este lenguaje crear modelos y ejecutarlos, cuando se deben desarrollar modelos extensos, de cientos o miles de líneas de código, las redes de íconos enlazados y distribuidos en múltiples pantallas se convierte en un problema que hace que estos modelos sean muy difíciles de gestionar.

Varias características hacen del GPSS/H de Wolverine una excelente opción para entornos de simulación. Una característica clave de GPSS/H es la *flexibilidad conceptual* para modelar un amplio rango de diferentes tipos de sistemas: cualquier sistema que pueda describirse como un flujo de procesos, con objetos y recursos que interactúan unos con otros, puede ser modelado [Crain99]. También provee *flexibilidad de especificación* dentro del lenguaje*:* fórmulas matemáticas complejas, expresiones y constantes pueden ser utilizadas virtualmente en cualquier lugar del modelo. Además, con muy poco esfuerzo, cualquier estadística automáticamente recolectada puede ser escrita en un archivo de salida para ser utilizada desde otro software.

La entrada de datos a los modelos GPSS/H puede realizarse tanto mediante el teclado al momento de su ejecución, como también desde archivos externos. Estos archivos no requieren un formato especial, ya que los valores en cada línea se pueden separar por espacios y *tabs*; GPSS/H posee funciones para leer estas líneas, detectar automáticamente valores enteros, caracteres y números en punto flotante de doble

precisión, y también permite especificar acciones al detectarse ciertas condiciones por la ocurrencia de un error o un EOF (*end of file*, fin de archivo).



```
Simulation begins.


Relative Clock: 2400.0000   Absolute Clock: 2400.0000


Block Current     Total  Block Current     Total
1                    15  11                  149
NEXTCYCL    4       164  12                  149
3                   160  13                    1
4           1       160  14                    1
5                   159
6                   159
7           8       159
8                   151
9           2       151
10                  149



         --Avg-Util-During--
 Storage  Total  Avail  Unavl     Entries    Average   Current  Percent    Capacity     Average     Current     Maximum
           Time   Time   Time              Time/Unit   Status   Avail                 Contents    Contents    Contents
  CRANES  0.955                       311      22.107    AVAIL   100.0           3       2.865           3           3


  Random    Antithetic     Initial     Current      Sample   Chi-Square
  Stream      Variates    Position    Position       Count   Uniformity
       1           OFF      100000      100634         634      0.70


Status of Common Storage

    7688 bytes available
    2312 in use
    2424 used (max)


Simulation complete.  Absolute Clock: 2400.0000


Total Block Executions: 1727

Blocks / second:        12193049

Microseconds / Block:   0.08


Elapsed Time Used (Sec)

Pass1:           0.00
Sym/Xref         0.00
Pass2:           0.00
Load/Ctrl:       0.00
Execution:       0.00
Output:          0.00
--------------------
Total:           0.00
gonetil@gonetil-laptop:~/.wine/drive_c/Wolverine/GPSSH$
```

Imagen 10: Visualización de resultados en GPSS/H

Para realizar múltiples ejecuciones de un mismo modelo, GPSS/H dispone de un lenguaje de *scripting* que permite construir experimentos y controlar la ejecución del modelo. Este lenguaje dispone de sentencias de control de interacciones (DO) y condicionales (IF-THEN-ELSE). Además de contar con este lenguaje de *scripting* embebido, la versión profesional de GPSS/H soporta rutinas externas escritas por el usuario en lenguaje C o FORTRAN. Si bien esto no suele ser necesario, puede resultar útil

para reutilizar código preexistente, o en caso de requerir rutinas computacionales extremadamente complejas, o incluso para generar interfaces de interoperabiilidad con otros softwares. Como se verá más adelante, esta práctica también es utilizada por GPSS World y el lenguaje PLUS. Finalmente, GPSS/H introdujo un conjunto de operaciones (SCANUCH y ALTERUCH) para operar con transacciones alojadas en USERCHAINs, que luego derivaron en los bloques SCAN y ALTER.

JavaGPSS

La habilidad de integración de GPSS/H con otros lenguajes de programación, si bien es útil, no es aplicable a todos los contextos. Más allá de su simple funcionalidad para manipular archivos, la coordinación con programas externos o simulaciones es bastante limitada. En particular, es posible incluir funciones externas e invocar programas externos, pero no permite la inclusión de la librería RTI (Runtime Infrastructure) [Klein97]. RTI es un *middleware*[6] (Imagen 11) requerido al implementar la Arquitectura de Alto Nivel (High Level Architecture, HLA). RTI es un componente fundamental de HLA. Provee un conjunto de servicios de software necesarios para soportar sistemas federados, coordinar sus operaciones e intercambiar datos durante la ejecución. Un sistema federado se compone de un conjunto de sistemas diferentes y autónomos, que son los participantes de esa federación. En primer lugar, estos participantes operan independientemente, pero tienen la posibilidad de ceder parte de su autonomía para participar de la federación [Busse99]. RTI es un implementación de la especificación de la interface HLA, pero no es en sí misma parte de la especificación. Las implementaciones modernas de RTI conforman las especificaciones de la IEEE 1516 [IEEE00] y/o la API HLA 1.3[SICS00][Möller04]. Estas especificaciones no incluyen un protocolo de red para RTI, y depende del implementador de cada RTI crear una especificación [Wiki13].

Para participar en una federación basada en HLA como un federado con todas las características, una herramienta debe ser capaz de cooperar con la RTI vía una interfaz de dos partes basada en una metáfora de un *ambassador.* El *ambassador RTI* es una

[6] Software que asiste a una aplicación para interactuar o comunicarse con otras aplicaciones, software, redes, hardware y/o sistemas operativos.

librería de software que puede ser enlazada a la herramienta o al modelo. Se incluye en la distribución de HLA y contiene métodos llamados por el federado. Por otro lado, el federado *ambassador* provee métodos llamados por el RTI y debe ser desarrollado e implementado como parte del proceso de modelado. La necesidad de implementar o incluir estos *ambassadors* presenta serios inconvenientes en la mayoría de las herramientas clásicas de simulación.

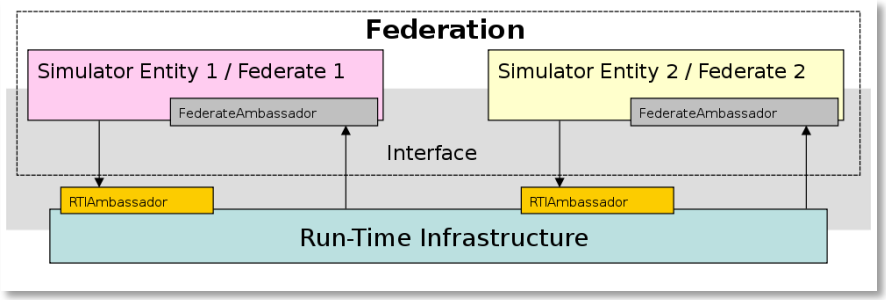


Imagen 11: Componentes de la especificación de RTI . Fuente: Wikipedia

A fin de proveer la funcionalidad necesaria y, al mismo tiempo, promover el uso de las ventajas de Internet, las herramientas de simulación y de animación de simulación son desarrolladas utilizando cada vez más el lenguaje Java. Una de estas herramientas se denomina Skopeo [Lorenz97][Dorwarth97], una herramienta de animación que lee archivos de trazas (*trace*) en un formato compatible con Proof Animation, una herramienta de Wolverine Software [Wolverine92], que provee animaciones básicas de simulaciones.

Por el lado de la simulación, una implementación en Java, denominada JavaGPSS, fue desarrollada por el ya mencionado Instituto para Simulación y Gráficos, de la Universidad Otto-von-Guericke, Magdeburgo (Alemania). JavaGPSS acepta sentencias en sintaxis GPSS, que luego son convertidas en código Java, compiladas en *bytecode* y ejecutadas en una máquina virtual (JVM) o entorno de ejecución (JRE) [Klein98]. Adicionalmente, JavaGPSS puede integrarse con HLA, lo que permite la ejecución de modelos GPSS extremadamente grandes en entornos distribuidos. Esta integración requiere considerar la gestión distribuida del tiempo de simulación, para la cual los autores del trabajo decidieron utilizar un protocolo de sincronización conservativa con *lookaheads* [Chen02], y la gestión de objetos de simulación distribuidos, que puede

realizarse a través del Simulation Object Model (SOM) y el Federation Object Model (FOM) definidos en el modelo de objetos por el HLA Object Model Template (HLA OMT, Imagen 12).

Finalmente, cabe mencionar que JavaGPSS no posee una interfaz gráfica de trabajo para el programador, ni tampoco provee de herramientas para mostrar ni analizar los resultados. La interacción se realiza, al igual que GPSS/H, mediante archivos de texto plano de entrada y de salida. Sin embargo, más allá de esta limitación, JavaGPSS representa un claro esfuerzo por llevar la potencia y flexibilidad de GPSS hacia entornos de ejecución más amplios, y así aplicar esta tecnología con modelos de simulación mucho más grandes y complejos.

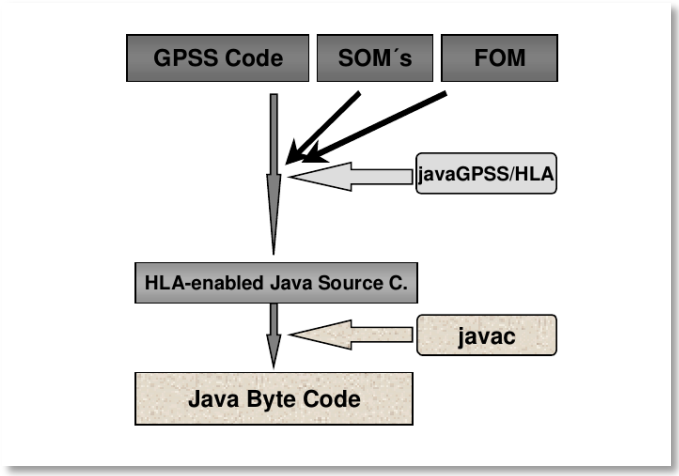


Imagen 12: Integración de código GPSS con HLA para su ejecución dentro de la JRE [Klein98]

GPSS World

GPSS World, de Minuteman Software[7], contiene una implementación del lenguaje GPSS, ampliada por un lenguaje embebido llamado PLUS (Programming Language Under Simulation) [Ståhl01]. Esta variante de GPSS incluye cerca de 50 bloques y 25 comandos, junto a 35 atributos numéricos del sistema (SNA). PLUS es un pequeño pero poderoso lenguaje de programación procedural creado con sólo 12 tipos de sentencias. Buena parte de su potencia proviene de su amplia librería de procedimientos, que contiene funciones matemáticas y para manipulación de *strings*, junto a un extenso conjunto de distribuciones de probabilidad. Esta versión de GPSS fue desarrollada exclusivamente

[7] Minuteman Software, página web: http://www.minutemansoftware.com.

para Windows. Si bien puede ejecutarse dentro de sistemas operativos de la familia GNU/Linux (mediante Wine), se observa una pérdida importante de estabilidad y una menor performance para ejecutar las simulaciones.

En GPSS World, un modelo se define como una secuencia de sentencias de modelo. Estas pueden ser sentencias GPSS (bloques y comandos), sentencias de procedimientos PLUS o sentencias de experimentos PLUS. Cualquier sentencia del modelo puede formar parte de un archivo con el modelo, o puede ser enviada interactivamente para modificar una simulación existente.

Una de las características más interesantes de GPSS World es su habilidad para ejecutar varias tareas a la vez mediante *multitasking*. El diseño basado en múltiples hilos de GPSS World le permite ejecutar múltiples simulaciones y experimentos de manera simultánea. No sólo la actualización de la pantalla, el ingreso de datos por parte del usuario, la entrada y salida desde el disco, la impresión y la simulación pueden realizarse concurrentemente, sino que pueden ejecutarse una gran cantidad de simulaciones al mismo tiempo. Estas funciones pueden hacer un uso muy intensivo de la memoria de la computadora, e incluso utilizar toda la memoria disponible. Sin embargo, las simulaciones no se limitan directamente al tamaño de la memoria física existente (RAM), sino que GPSS gestiona su propia memoria virtual en disco. Si bien esto puede provocar retrasos en la velocidad de ejecución de las simulaciones, dada la diferencia de los tiempos de acceso entre la memoria RAM y el disco, este mecanismo permite ejecutar simulaciones muy grandes, e incluso varias de éstas de manera concurrente.

En el nivel de la interfaz, GPSS World es una implementación de la arquitectura Document/View, común en las aplicaciones Windows. Los objetos pueden ser abiertos con múltiples vistas, modificados y almacenados en memoria persistente de acuerdo a secuencias de acciones muy intuitivas para el usuario. Gracias a esto, GPSS World posee más de 20 ventanas diferentes (Imagen 13), que le permiten al usuario conocer el estado de cada una de las entidades que ha utilizado. Estas ventanas pueden ser accedidas al finalizar la simulación, pero también durante la ejecución de la misma, lo que permite observar cómo va cambiando cada entidad a medida que la simulación avanza. Si bien esto último puede hacer considerablemente más lenta la ejecución de la simulación, es de gran utilidad para *debuggear* modelos grandes o analizar el estado de la simulación

en un punto intermedio. La información que muestran estas ventanas es muy útil para usuarios avanzados de GPSS, pero no ayudan al usuario principiante a comprender cómo funciona el motor de simulación, o cómo se relacionan los distintos elementos de GPSS. Esto se debe principalmente a que los datos de las entidades mostrados en estas ventanas se encuentran aislados de otras entidades, y no es posible acceder a una entidad desde la ventana de otra. Por ejemplo, al visualizar una facilidad, se muestra la cantidad de transacciones en cada una de sus cadenas, al igual que el reporte, pero no se puede acceder a este listado de transacciones. Otro ejemplo, al mostrar la FEC, se pueden ver las transacciones, junto a su bloque actual, pero no se puede acceder a dicho bloque. En realidad, cada una de estas ventanas muestra la misma información que se genera en el reporte final.

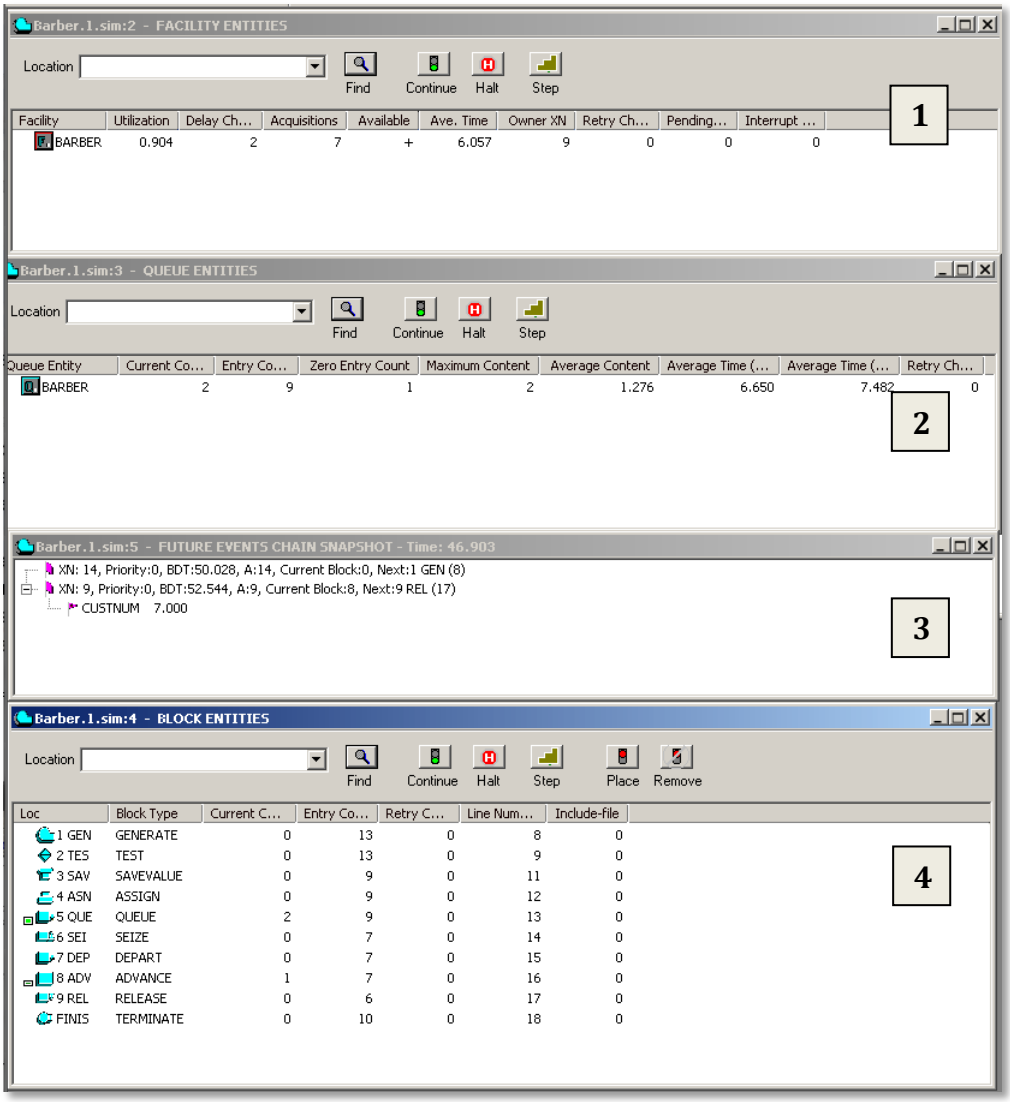


Imagen 13: De arriba a abajo: 1) ventana de vista de facilidades; 2) ventana de vista de colas; 3) ventana de vista de la FEC y 4) ventana de vista de bloques

Para la creación de modelos, GPSS World presenta al programador un editor de texto plano, similar al Notepad de Windows (Imagen 14). Este editor no brinda asistencia alguna a los usuarios, pues está pensado para usuarios expertos que conocen la sintaxis de GPSS y las entidades con las que cuentan. Los modelos son ejecutados mediante el comando START, y el resultado de la simulación se presenta en un reporte final (*report,* Imagen 15) que resume el estado de todas las entidades permanentes, el paso de las transacciones por la lista de bloques, y muestra algunos datos globales como, por ejemplo, la duración de la simulación, la cantidad de bloques, de facilidades y de almacenes, la fecha y hora, entre otros. Este reporte también puede incluir el listado completo de transacciones remanentes tanto en la FEC como en la CEC al momento de finalizar la simulación.

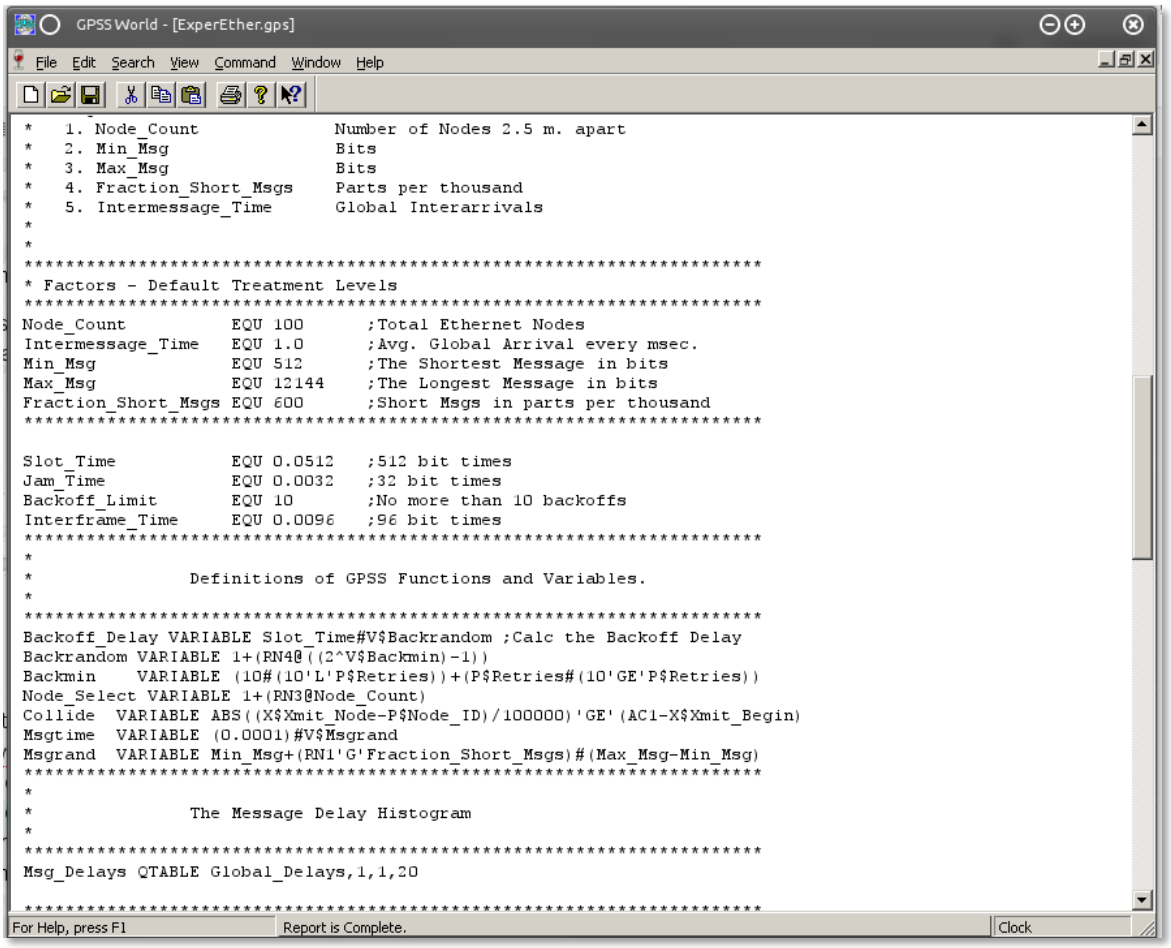


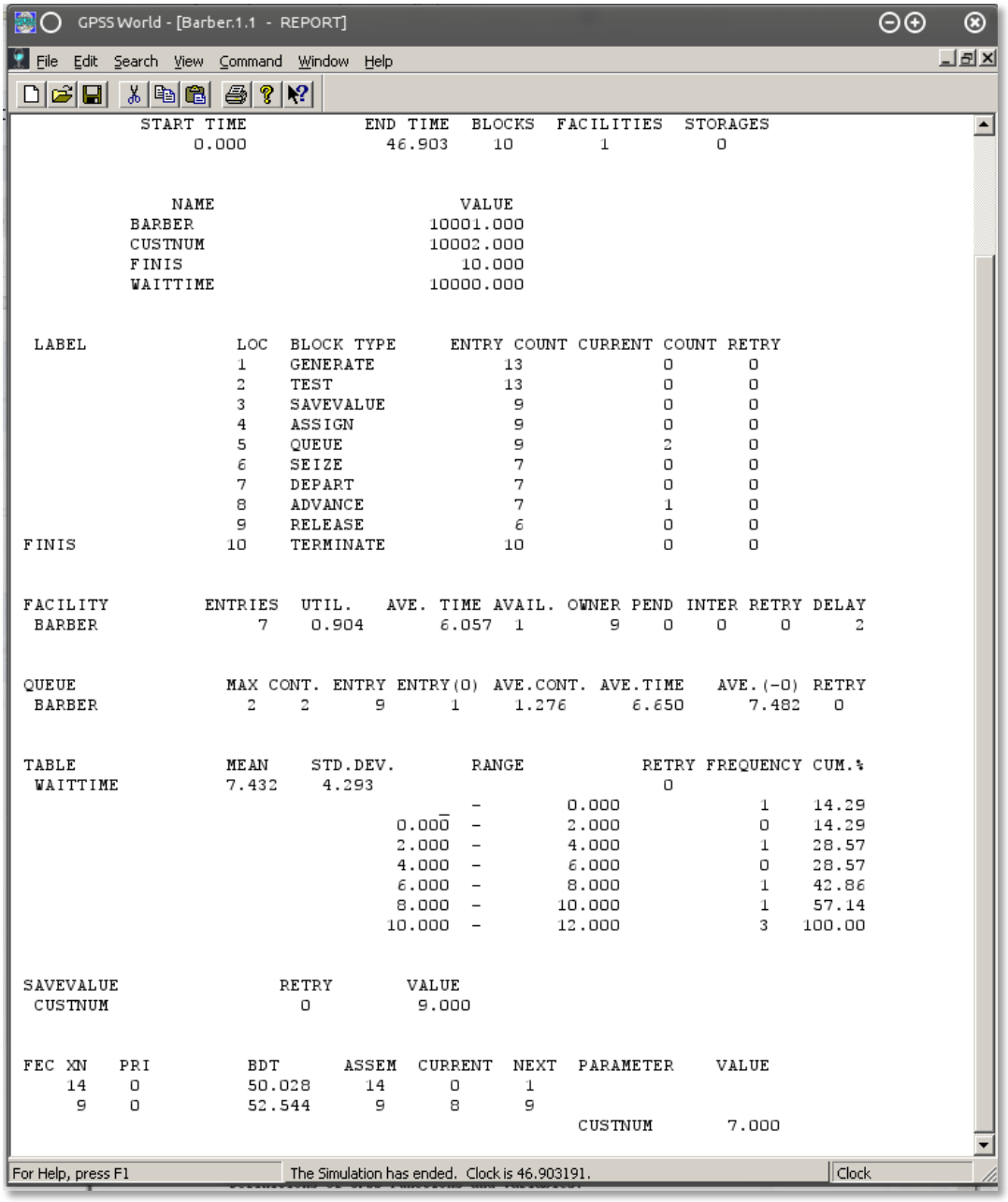
Imagen 14: Editor de código de GPSS World

Conclusiones

En este capítulo se analizaron varios estudios relativos al diseño de entornos de enseñanza interactivos en general, la enseñanza en el área de sistemas discretos, y los requisitos que deben tener las herramientas de enseñanza interactiva en el área. El diseño del sistema de ayuda y asistencia en los entornos de aprendizaje cobra un papel muy importante aquí. Este sistema debe proveer información útil en el momento adecuado, con un nivel de abstracción que debería ir en aumento a medida que el estudiante adquiere mayores habilidades cognitivas. Si bien, en un principio, la información brindada puede ser simple y concreta, los estudiantes deben aprender a buscar ayuda e interpretar la información encontrada. Esto indica que la búsqueda de ayuda debe pasar paulatinamente a manos de los estudiantes, y el sistema debe proveer información suficiente para orientarlos a la hora de cumplir con una actividad. Para brindar asistencia, el entorno de aprendizaje puede apoyarse en métodos directos, como enlaces a documentos, diálogos, ayudas contextuales y mensajes de error, y también a métodos indirectos o sutiles, como imágenes, iconos, mecanismos de interacción o incluso aspectos de diseño de la interfaz visual. Adicionalmente, los entornos de enseñanza interactivos deben diseñarse pensando en cómo se aplicarán en clase, cómo se integrarán con las actividades que realizarán los alumnos, en qué momento lo harán y de qué manera.

En lo relativo a la enseñanza en el área de simulación de eventos discretos, existen diversos factores como la cantidad y diversidad de temas, y la experiencia y el conocimiento previo de los estudiantes, que hacen que a la hora de planificar el curso de DES en general así como también cada uno de los encuentros, se requiera una importante flexibilidad por parte del docente para adaptar los contenidos a enseñar y las actividades a realizar con los estudiantes. La herramienta de software utilizada para simulación debe servir como apoyo al curso, pero no es el objetivo del curso aprender a utilizar esta herramienta. Se considera muy valioso que esta herramienta permita relacionar de manera natural los temas teóricos con los conceptos prácticos. También es necesario que el docente logre que los alumnos se interesen en el aprendizaje, pues de esto dependerá que estos futuros profesionales sepan aprovechar toda la potencia de la

simulación como herramienta de análisis de sistemas en su vida profesional. Tanto el diseño del curso, como el rol de los estudiantes y las herramientas de software utilizadas son factores cruciales que permiten aumentar el interés de los alumnos.



GPSS World - [Barber.1.1 - REPORT]

File Edit Search View Command Window Help

```
          START TIME          END TIME  BLOCKS  FACILITIES  STORAGES
               0.000            46.903    10        1          0


            NAME                    VALUE
         BARBER                   10001.000
         CUSTNUM                  10002.000
         FINIS                       10.000
         WAITTIME                 10000.000


 LABEL              LOC  BLOCK TYPE     ENTRY COUNT CURRENT COUNT RETRY
                    1    GENERATE            13              0       0
                    2    TEST                13              0       0
                    3    SAVEVALUE            9              0       0
                    4    ASSIGN               9              0       0
                    5    QUEUE                9              2       0
                    6    SEIZE                7              0       0
                    7    DEPART               7              0       0
                    8    ADVANCE              7              1       0
                    9    RELEASE              6              0       0
FINIS               10   TERMINATE           10              0       0


FACILITY         ENTRIES  UTIL.   AVE. TIME AVAIL. OWNER PEND INTER RETRY DELAY
 BARBER              7    0.904       6.057  1        9    0    0     0       2


QUEUE              MAX CONT. ENTRY ENTRY(0) AVE.CONT.  AVE.TIME   AVE.(-0) RETRY
 BARBER              2    2      9       1     1.276      6.650      7.482   0


TABLE              MEAN    STD.DEV.       RANGE            RETRY FREQUENCY CUM.%
 WAITTIME          7.432    4.293                            0
                                      -        0.000             1    14.29
                                0.000 -        2.000             0    14.29
                                2.000 -        4.000             1    28.57
                                4.000 -        6.000             0    28.57
                                6.000 -        8.000             1    42.86
                                8.000 -       10.000             1    57.14
                               10.000 -       12.000             3   100.00


SAVEVALUE               RETRY       VALUE
 CUSTNUM                  0          9.000


FEC XN   PRI         BDT      ASSEM  CURRENT  NEXT  PARAMETER    VALUE
    14    0          50.028     14      0      1
     9    0          52.544      9      8      9
                                                  CUSTNUM       7.000
```

For Help, press F1 | The Simulation has ended. Clock is 46.903191. | Clock

Imagen 15: Reporte de GPSS World

Se analizaron también algunas herramientas y entornos de enseñanza diseñados para el aprendizaje de GPSS. Una aproximación común en varios desarrollos fue el uso de

diálogos para insertar sentencias GPSS en el modelo, en particular bloques y sus parámetros. Además, muchos entornos recurren también a una representación gráfica de la red de bloques: a cada bloque se le asigna un símbolo, y el usuario interactúa directamente con estos símbolos en vez de ver código GPSS estándar. La interacción se realiza mediante eventos con el ratón: clic, arrastrar y soltar. El uso de sistemas multiplataforma, y en especial de sistemas basados en entornos web, presenta muchas ventajas aquí. Sin embargo, muy pocas implementaciones son realmente multiplataforma (en particular, aquellas desarrolladas en lenguaje Java: JGPSS y JavaGPSS), y sólo una —WebGPSS— fue desarrollada para entornos web, aunque de hecho se trata de una adaptación de la misma aplicación para entornos de escritorio (WinGPSS) que se ejecuta dentro de un *applet* Java en el navegador. FONWebGPSS es un caso particular aquí, pues integra el desarrollo de modelos en GPSS con el entorno de enseñanza Moodle. Esta integración no incluye un sistema interactivo para la construcción de modelos, ni sistemas de ayuda embebidos, sino que se limita a proveer un área de texto plano para que el estudiante escriba el código GPSS, lo envíe al servidor para ejecutarse, y visualice los resultados en el mismo navegador.

Una propuesta interesante es la presentada con el lenguaje ISDS, que busca aprovechar todo el conocimiento previo que los estudiantes poseen de lenguajes de programación estructurada, al estilo Pascal o C. Este lenguaje posee sentencias con nombres similares a GPSS, pero los modelos se estructuran mediante sentencias procedurales que incluyen asignaciones, condicionales, repeticiones y definición explícita de entidades. Luego, el compilador de ISDS traduce el código procedural al modelo GPSS, y lo ejecuta dentro de un motor de simulación propio. Los resultados de este trabajo indican que este lenguaje permite lograr un ahorro de tiempo; sin embargo, una vez finalizado el curso los alumnos deben aprender a utilizar el lenguaje GPSS real.

Más allá de las características de la herramienta utilizada a lo largo del curso de simulación de eventos discretos, se observó que los distintos trabajos reseñados se limitaron a presentar las principales características de cada herramienta pero, a diferencia de lo que sugieren los trabajos de Arrollo, de Aleven y de Woods, no fueron acompañados por un diseño del curso orientado específicamente para aprovechar lo mejor de cada herramienta. Por el contrario, en estos trabajos los autores utilizaron

estas herramientas de enseñanza bajo el esquema tradicional del curso de DES. Esta es una diferencia fundamental con la propuesta de esta tesis, en la que se presenta una herramienta de enseñanza junto a una metodología que permite aprovechar mejor las ventajas de esta herramienta.

Finalmente, se analizaron las principales implementaciones profesionales de GPSS. Todas las herramientas presentan características muy avanzadas, y en algunos casos han sido diseñadas para determinados contextos específicos, como por ejemplo la integración de JavaGPSS con HLA para simulaciones distribuidas o GPSS World para sistemas con soporte *multithread*. Estas herramientas están pensadas para ser usadas por expertos, y ofrecen una gran variedad de funciones, tecnologías y sistemas de análisis de resultado. Sin embargo, en ningún caso fueron pensadas con fines pedagógicos: no disponen de asistentes o sistemas de ayuda contextuales, no están pensadas para integrarse con un curso de simulación de eventos discretos, y sus interfaces avanzadas —o falta de interfaz— no son adecuadas para estudiantes que están comenzando sus estudios en esta área.

Capítulo 3

Simulación y GPSS

La simulación por computadora es una técnica para predecir cómo sistemas, nuevos o alterados, se comportarán bajo ciertas condiciones. Esta técnica es mucho más utilizada que los métodos puramente matemáticos, ya que permite manejar sistemas mucho más complejos, y su uso correcto permite a las empresas ahorrar tiempo y dinero. La mayoría de los sistemas del mundo real pueden ser simulados, por lo general, a partir de modelos abstractos y de alguna herramienta en particular.

De los lenguajes de simulación desarrollados hasta la fecha, ninguno ha provocado mayor impacto que el Sistema de Simulación de Propósito General, o GPSS por sus siglas en inglés (General Purpose Simulation System). Este sistema fue desarrollado por Geoffrey Gordon, de IBM, a comienzos de los sesenta, y es en la actualidad uno de los lenguajes de simulación más populares. Su diseño ha probado ser tan robusto que, al día de hoy, aún provee las bases para el desarrollo de entornos de simulación modernos, y ha influenciado muchos otros lenguajes que ahora se basan en derivaciones de los conceptos de GPSS.

Este capítulo tiene como objetivo, en su primera parte, introducir de manera general la terminología fundamental y los conceptos centrales del área de modelado y simulación por computadoras, a fin de brindar un panorama general sobre los temas de los que trata este trabajo así como también el curso de Modelos y Simulación. La segunda parte, se enfoca sobre el Sistema de Simulación de Propósito General (GPSS) en particular, no sólo desde el punto de vista del lenguaje y del motor de simulación, sino también haciendo énfasis sobre algunos aspectos particulares desde el punto de vista de la enseñanza. Cabe destacar que esta sección no tiene como objetivo abarcar la totalidad de los conceptos de Modelado y Simulación, ni tampoco cubrir todas las características y entidades de GPSS, ya que esta información puede encontrarse en muchos libros y

tutoriales en línea, y extendería demasiado este texto sin realizar un aporte concreto.

Introducción a la simulación

Simular es imitar la operación de un proceso o sistema del mundo real durante un tiempo determinado. Ya sea que se realice con una computadora o sin ella, la simulación involucra la generación de una historia artificial de un sistema, y la observación de esa historia artificial para extraer inferencias sobre las características operativas del sistema real.

El comportamiento de un sistema del mundo real durante un cierto tiempo se estudia desarrollando un modelo de simulación. Este modelo, por lo general, contiene una serie de asunciones o abstracciones en lo relativo a la operación del sistema, las cuales son expresadas en términos de relaciones simbólicas, matemáticas o lógicas entre entidades (objetos de interés) del sistema. El desarrollo de un modelo de simulación implica una etapa de validación, que sirve para verificar que el modelo cumple con la operación y las restricciones del sistema real. Una vez que fue desarrollado y validado, el modelo puede ser utilizado para investigar una amplia variedad de preguntas del tipo "qué pasaría si..." sobre el sistema de mundo real. Los cambios potenciales sobre el sistema pueden primero ser simulados para predecir su impacto en la performance del sistema. La simulación puede también utilizarse para estudiar sistemas en su etapa de diseño, antes de que tales sistemas sean construidos. De este modo, los modelos de simulación pueden ser utilizados como herramienta de análisis para predecir el efecto de los cambios sobre sistemas existentes, y también como herramienta de diseño para predecir la performance de nuevos sistemas bajo un conjunto cambiante de circunstancias [Banks84].

En algunos casos, los modelos pueden desarrollarse de manera lo suficientemente "simple" como para resolverse mediante métodos matemáticos. Tales soluciones pueden encontrarse por el uso de cálculo diferencial, teoría de la probabilidad, métodos algebraicos o técnicas matemáticas. La solución, por lo general, consta de uno o más parámetros numéricos, que son llamados *medidas de performance del sistema.* Sin embargo, muchos sistemas del mundo real son demasiado complejos, y los modelos para

tales sistemas son virtualmente imposibles de resolver matemáticamente. En estas circunstancias, la simulación numérica basada en computadoras puede ser utilizada para imitar el comportamiento del sistema a lo largo del tiempo. A partir de la simulación, los datos son recolectados como si se estuviera observando el sistema real. Estos datos generados por la simulación son utilizados para *estimar* las medidas de performance del sistema.

¿En qué casos la simulación es la herramienta apropiada?

La disponibilidad de los lenguajes de simulación de propósito específico, las capacidades de computación masiva a un costo en continuo decrecimiento y los avances en las metodologías de simulación, han hecho de la simulación una de las herramientas más usadas y aceptadas en operaciones de investigación y análisis de sistemas. Sin embargo, debe considerarse si las circunstancias en las cuales se aplicará esta técnica son las más apropiadas. La simulación puede utilizarse en numerosas situaciones:

- Estudiar o experimentar sobre las interacciones internas de un sistema complejo, o de un subsistema dentro de un sistema complejo.
- Los cambios organizacionales y ambientales pueden ser simulados, y el efecto de dichas alteraciones sobre el comportamiento del modelo puede ser observado.
- El conocimiento ganado durante el diseño de un modelo de simulación puede ser de gran valor para sugerir mejoras en el sistema bajo estudio.
- Mediante ciertos cambios en los parámetros de entrada de la simulación, y la observación de los resultados de salida, puede obtenerse un conocimiento muy valioso sobre cuáles variables son más importantes y sobre cómo interactúan unas con otras.
- La simulación puede utilizarse como un dispositivo pedagógico para reforzar metodologías de solución analíticas.
- La simulación puede aplicarse para experimentar con nuevos diseños, métodos o políticas antes de su implementación, y así preparar el sistema para lo que podría venir.
- Las soluciones analíticas pueden verificarse mediante simulación.

El uso de la simulación, como de cualquier otra herramienta, requiere una previa consideración sobre sus ventajas y desventajas antes de aplicar la metodología. A continuación, se listan algunas de las desventajas e inconvenientes del uso de la simulación [Arias03][Dunna06]:

- ✓ El desarrollo de un modelo de simulación tiene un costo asociado, que se incrementa a medida que el sistema se hace más complejo y a medida que se desea controlar una mayor cantidad de *variables* del modelo. Al tiempo de desarrollo debe sumarse también el esfuerzo necesario para validar el sistema y realizar los ajustes necesarios cuando la validación no es satisfactoria.
- ✓ Los modelos de simulación requieren por lo general numerosas *corridas* o *ejecuciones*. A pesar del avance en la capacidad de cálculo de las computadoras, algunos modelos de simulación pueden ser extremadamente complejos y tomar demasiado tiempo para su ejecución.
- ✓ Cuando las técnicas analíticas son suficientes, no es necesario construir y validar un modelo de simulación. Esto suele suceder cuando los usuarios se sienten cómodos con la metodología de simulación, y dejan de lado los métodos matemáticos.
- ✓ Cuando se desarrollan modelos estocásticos, es posible observar diferentes comportamientos en el sistema al ejecutarlos. Esta variabilidad debe ser considerada por el analista al momento de estudiar alternativas de optimización del sistema.
- ✓ La calidad del estudio estará acotada por la calidad del modelo, al igual que ocurre en cualquier modelado de sistemas. Si el modelo desarrollado no representa el sistema de manera suficientemente aproximada, las conclusiones inferidas de los resultados de las simulaciones pueden ser incorrectas. De aquí la importancia de validar el modelo.

Más allá de estas desventajas, la simulación presenta un amplio conjunto de ventajas entre las que se destacan:

- Una vez que el modelo ha sido construido, puede utilizarse repetidamente para analizar diseños o políticas propuestas. El reuso de modelos es un factor determinante al momento de aplicar esta técnica. A medida que el analista adquiere mayor capacidad y conocimiento con el uso de esta herramienta, es capaz de realizar modelos más abstractos y genéricos, que facilitan su reuso bajo diferentes circunstancias.

- La simulación permite estudiar sistemas en un marco temporal adecuado, comprimiendo el tiempo o expandiéndolo según sea el caso. El tiempo se contrae cuando la evolución del sistema es muy lenta, y se dilata cuando es demasiado rápida.

- Los métodos de simulación pueden ser utilizados para ayudar a analizar un sistema propuesto incluso cuando los datos de entrada son de algún modo inseguros o cuestionables.

- Una simulación permite mantener un mejor control sobre las condiciones de funcionamiento que el que se obtiene experimentando directamente sobre el sistema real. Además, en muchos casos, la experimentación sobre el sistema real no es una alternativa viable, por lo general por cuestiones de costos y tiempos.

- La simulación es por lo general más fácil de aplicar que los métodos analíticos. Por lo tanto, existen más usuarios potenciales de los modelos desarrollados.

- Los métodos analíticos requieren por lo general muchas suposiciones y simplificaciones para hacerlos matemáticamente controlables y manejables; sin embargo, los modelos de simulación no poseen tales restricciones. Con los métodos analíticos, el analista por lo general puede computar sólo un limitado número de medidas de performance del sistema. Con los modelos de simulación, los datos generados pueden ser utilizados para estimar cualquier medida de performance concebible.

Sistema y modelo

Un sistema puede definirse como una colección de entidades que actúan e interactúan para la consecución de un determinado fin. Los elementos de un sistema se relacionan para funcionar como un todo; desde el punto de vista de la simulación, tales elementos

deben tener una frontera clara. Los objetivos del estudio del sistema condicionan, por lo general, el conjunto de entidades que se consideran; para ciertos estudios puede ser suficiente un subconjunto de entidades del sistema global. En otras palabras, lo que constituye el sistema es relativo al foco del estudio y la extensión deseada de las conclusiones [Biles 87].

Cada entidad es un objeto de interés en el sistema. Estas entidades poseen propiedades, también llamadas atributos. Una actividad representa un período de tiempo de una duración específica. Por ejemplo, si se está estudiando un banco, los clientes podrían ser las entidades, el balance entre sus cuentas podría ser un atributo, y la realización de un depósito podría ser una actividad.

El estado de un sistema se define como la colección total de variables necesarias para describir el sistema en cualquier instante de tiempo, relativo a los objetivos del estudio. Tales variables son conocidas como *variables de estado*. La evolución temporal de las variables de estado de un sistema permite establecer una clasificación de los tipos de sistemas:

- Sistemas discretos, en los cuales las variables de estado cambian en un conjunto de instantes de tiempo contable, que puede ser finito o infinito numerable.
- Sistemas continuos, en los cuales las variables de estado cambian de manera continua a lo largo del tiempo.

Pocos sistemas en la práctica son completamente discretos o continuos, pero dado que un tipo de cambio predomina para la mayoría de los sistemas, por lo general es posible clasificarlos o bien como discretos, o bien como continuos [Law00].

Un modelo es una representación de un sistema. Esta representación tiene como objetivo el estudio de dicho sistema para comprender las relaciones entre sus componentes o para predecir cómo el sistema operará bajo nuevas políticas. Se construyen modelos cuando no es posible o conveniente experimentar con el sistema real, y también cuando el sistema aún no existe, ya que podría tratarse de un sistema hipotético o estar en una etapa de diseño. Para la mayoría de los estudios no es necesario considerar todos los detalles del sistema. Por lo tanto, un modelo no es sólo un sustituto del sistema, es también una simplificación del mismo, una interpretación útil a

determinados fines [Mihram74]. Sin embargo, el modelo debería ser lo suficientemente detallado para permitir realizar conclusiones válidas sobre el sistema. Claramente, un sistema puede requerir múltiples modelos, según cambien los propósitos de la investigación. Por ejemplo, durante la etapa de construcción de una casa, se generan diversos modelos que representan diferentes aspectos de la obra en construcción:

- planos bidimensionales, que representan la distribución de los ambientes y espacios de la casa, accesos, escaleras, etc.;
- maquetas tridimensionales, que sirven para estudiar el flujo de aire y luz en la casa;
- representaciones por computadora de los ambientes, para analizar distintas combinaciones de colores de paredes, aberturas y techos;
- diagramas que muestren las distintas instalaciones eléctricas de la vivienda (cableados, puntos de contacto, tomacorrientes, interruptores magnetotérmicos, etc.).

Todos los modelos arriba mencionados describen el mismo sistema global (vivienda), pero se centran en aspectos muy diferentes y, por lo tanto, las entidades que se incluyen en cada modelo variarán notoriamente de las entidades de los otros modelos.

El modelado y simulación es fundamentalmente un proceso interactivo. Típicamente, el analista construye un modelo del sistema, diseña un experimento, ejecuta la simulación utilizando dicho experimento, y observa la salida. Basado en los objetivos del estudio, el analista puede realizar ajustes al modelo y al experimento, y volver a ejecutar la simulación. Muchos lenguajes proveen algún tipo de soporte para asistir al analista durante la etapa de modelado. Por ejemplo, EMSS [Ford87], EZSIM [Khoshnevis87] y KBMC [Murray88] cuentan con un sistema experto para asistir al analista en una o más fases del proceso de modelado [Pollacia89].

Los distintos tipos de modelos se clasifican en dos grandes categorías:

- ✓ *Modelos matemáticos*, que utilizan notación simbólica y ecuaciones matemáticas para presentar un sistema. Los modelos de simulación son un caso particular de modelos matemáticos.

- ✓ *Modelos físicos*, muy utilizados en las industrias aeronáutica y automotriz, en las cuales se desarrollan modelos a escala para realizar pruebas directamente sobre ellos.

Los modelos de simulación pueden también clasificarse como estáticos o dinámicos, determinísticos o estocásticos, y discretos o continuos. Un modelo de simulación estático, también conocido como una simulación Monte Carlo, representa un sistema en un punto particular del tiempo. Los modelos de simulación dinámicos, por el contrario, representan sistemas mientras cambian a lo largo del tiempo.

Según la aleatoriedad de las variables de estado, los modelos de simulación pueden ser determinísticos y estocásticos. Los modelos determinísticos no contienen variables aleatorias, y sus sucesivas ejecuciones ante un mismo conjunto de valores de entrada provocará siempre el mismo conjunto de valores de salida. Los modelos de simulación estocásticos poseen una o más variables aleatorias como entradas. Las entradas aleatorias llevan a salidas aleatorias. Dado que las salidas son aleatorias, pueden ser consideradas sólo como estimaciones de las verdaderas características de un modelo. De aquí que, en una simulación estocástica, las medidas de salida deben ser tratadas como estimaciones estadísticas de las verdaderas características del sistema en estudio.

De acuerdo al modo en que evolucionan las variables de estado, los modelos de simulación pueden ser discretos, si sus variables de estado varían en un conjunto contable de instantes de tiempo, o continuos si las variables de estado varían de modo continuo en función del tiempo. Sin embargo, un modelo de simulación discreto no siempre es usado para modelar un sistema discreto, y tampoco un modelo de simulación continuo es usado para modelar sistemas continuos. La opción de cuándo usar un sistema continuo, uno discreto o ambos (modelos combinados) dependerá de las características del sistema y el objetivo del estudio.

En el contexto de este trabajo, los modelos considerados son **discretos, dinámicos y estocásticos**.

Simulación de eventos discretos

La simulación de eventos discretos es el modelado de sistemas en los cuales la

variables de estado cambian solo en un conjunto discreto de puntos en el tiempo [Nance81]. Los modelos de simulación son analizados por métodos numéricos en vez de métodos analíticos. Los métodos analíticos emplean el razonamiento deductivo de las matemáticas para "resolver" el modelo. Por ejemplo, el cálculo diferencial puede ser usado para determinar la política de costo mínimo para cierto modelo de inventario. Los métodos numéricos emplean procesos computacionales para "resolver" modelos matemáticos. En los modelos de simulación, se dice que los modelos son ejecutados en vez de resueltos. Durante la ejecución, se genera una historia artificial del sistema basada en las asunciones del modelo, y se recolectan observaciones que luego serán analizadas y estimadas sobre las medidas de performance del sistema real. Dado que los modelos de simulación reales son por lo general grandes, y dado que la cantidad de datos que éstos generan y manipulan suele ser muy amplia, estas corridas se realizan en general con ayuda de las computadoras. Más allá de esto, los modelos de simulación pequeños pueden ser ejecutados a mano, y también permiten obtener información muy valiosa para el analista.

Simulación de sistemas gobernados por colas

Un sistema gobernado por colas se describe a partir de la población que lo accede, la naturaleza de las llegadas y los servicios, la capacidad del sistema, y la disciplina de encolamiento.

Por lo general, la población que accede al sistema se considera infinita; eso significa que, si una unidad deja el conjunto total de la población y se une a la cola de espera o ingresa a un servicio, no se modifican las tasas de arribos de las unidades que pueden requerir servicios. Otra característica de estos sistemas es que los arribos en busca de servicios ocurren uno a la vez en forma aleatoria y, una vez que se unen a una cola de espera, serán servidos en algún momento. Además, los tiempos de servicio poseen por lo general cierta aleatoriedad de acuerdo a distribuciones de probabilidad que no cambian a lo largo del tiempo. Se considera también que el sistema posee capacidad ilimitada, considerando al sistema como la unidad siendo servida más todas las unidades en cola de espera. Por último, las unidades o usuarios son servidos (o atendidos) en orden de llegada o FIFO (*first in, first out*) por un único servidor o canal.

Las llegadas y los servicios se describen en términos de las distribuciones de tiempo entre arribos y tiempos de servicio. La tasa global de llegadas efectivas debe ser menor a la tasa de servicio máxima, o de lo contrario la cola de espera crecerá sin límites; eso se conoce como sistema explosivo o inestable.

En los sistemas gobernados por colas, el estado del sistema está dado por el número de unidades en el sistema y por el estado del servidor (libre u ocupado). Un *evento*, en este contexto, es un conjunto de circunstancias que causan un cambio instantáneo en el estado del sistema. Un evento inicia una *actividad*, que tiene una longitud de tiempo durante la cual una o más entidades se comprometen en alguna operación [Pollacia89]. El tiempo de la actividad es conocido al momento de comenzar la actividad, aunque puede obtenerse a partir de una función estadística [Banks84]. Un *proceso* es una secuencia de eventos que pueden involucrar varias actividades juntas. La imagen 16 ilustra la relación entre evento, actividad y proceso.

En un sistema de colas de canal simple existen sólo dos posibles eventos que pueden afectar el estado del sistema: el ingreso de una unidad al sistema (evento de arribo o llegada) y la finalización de un servicio en una unidad (evento de partida o salida). El sistema de encolamiento incluye al servidor, la unidad que está siendo servida (si es que hay alguna), y las unidades en cola de espera (si es que existe alguna esperando).

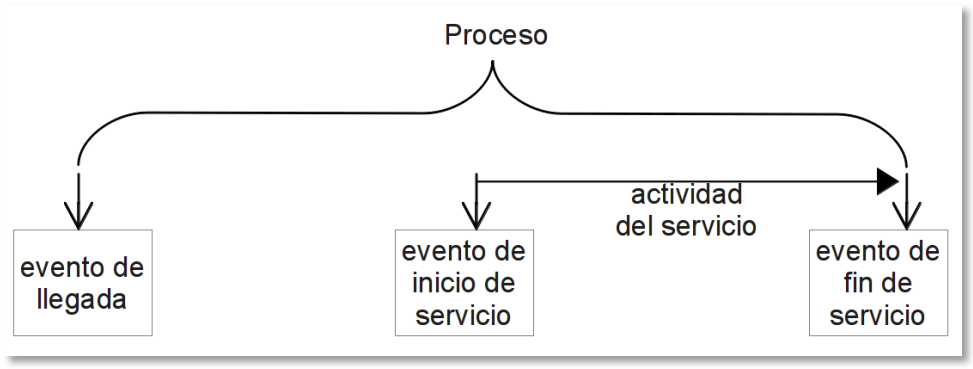


Imagen 16: Relación entre Proceso, Evento y Actividad

Al completarse un servicio, el sistema analiza si existe alguna unidad en espera. En caso negativo, el servidor pasa al estado libre (*idle*); en caso positivo, la primera unidad (en orden FIFO) sale de la cola de espera, y el servidor comienza a servirla, en cuyo caso permanece en estado ocupado (*busy*). Es importante destacar que el servidor sólo puede

estar libre u ocupado, o sea que no existe otro estado posible.

El segundo evento sucede cuando una unidad ingresa al sistema. Ante tal evento, el sistema analiza el estado del servidor: si el servidor estaba libre, la unidad ingresa al mismo para ser servida y el servidor pasa a estado ocupado; si el servidor estaba ocupado, la unidad ingresa al final de la cola de espera del servidor. Nunca puede suceder que el servidor esté libre y que haya unidades en la cola de espera. Tampoco puede suceder que, una vez completado un servicio, el servidor permanezca ocupado si no hay unidades en la cola de espera.

Durante la simulación de sistemas gobernados por colas, por lo general se requiere mantener una lista de eventos a futuro, que determinan qué sucederá a continuación. Esta lista de eventos indica los tiempos en los cuales los diferentes tipos de eventos ocurrirán para cada unidad en el sistema. Los tiempos se mantienen sobre un "reloj interno" [Neelamkavil87], que se encarga de marcar las ocurrencias de eventos a lo largo del tiempo. En simulación, los eventos suelen ocurrir de manera aleatoria, lo cual intenta imitar la incertidumbre de la vida real.

La aleatoriedad requerida para imitar la vida real es posible gracias al uso de números aleatorios, los cuales son distribuidos de manera uniforme e independiente en el intervalo (0,1). Los números aleatorios son generados utilizando un procedimiento, y son comúnmente llamados números pseudo-aleatorios ya que el método de generación es conocido y, por lo tanto, siempre es posible conocer cuál será la secuencia de números antes de comenzar la simulación [Park88][Robert04].

Lenguajes de simulación

En cualquier estudio de simulación por computadora existen dos etapas distintas: primero, la de análisis de sistemas, en la cual se define el modelo de simulación; y segundo, la programación en computadora, en la cual se codifica el modelo para procesar los datos.

Como ya se mencionó previamente, cada entidad de los sistemas posee ciertas características o atributos que pueden ser usados por el analista para describir detalles de su función en el modelo. Mediante la definición de un conjunto apropiado de tales

entidades del modelo, los esfuerzos de programación pueden verse reducidos drásticamente. La totalidad de los elementos de modelado forma un lenguaje de programación de simulación, como por ejemplo el lenguaje GPSS del que trata este trabajo. Para ser efectivo, un lenguaje de simulación debe ser capaz de manejar un conjunto mínimo de funciones con poco esfuerzo por parte del usuario[Bobillier76]:

- *Tiempo*: la simulación es dinámica; el modelador necesita por lo tanto examinar el comportamiento del sistema con tiempo. Debe proveerse un mecanismo para que el modelador pueda introducir retrasos en el modelo y para registrar el tiempo simulado.
- *Eventos*: el modelo lidiará con el tiempo continuo de un sistema real, pero para economizar la cantidad de instrucciones a ejecutar, el modelo cambiará sólo cuando suceda un cambio significativo en el estado del sistema. El lenguaje de modelado debe ser capaz de manejar todos los eventos en un orden lógico. En otras palabras, los numerosos eventos que tendrán lugar durante cualquier simulación deben ser secuenciados correctamente, sujetos a condiciones de prioridades si existieran, y debe también manejar la congestión que puede alterar el esquema de planificación originalmente planeado.
- *Variables aleatorias*: el lenguaje de simulación debe proveer técnicas de generación de números aleatorios eficientes, para poder ejecutar modelos estocásticos en la computadora.
- *Recolección y reporte de estadísticas*: el simulador debería recolectar algunas estadísticas mínimas de manera automática, y proveer al analista algunos métodos simples para especificar y recolectar información de importancia en el modelo actual. Es asimismo necesario contar con una salida estándar generada cuando la ejecución de la simulación finaliza, ya sea bajo condiciones normales o ante la ocurrencia de un error. Es deseable también contar con la posibilidad de registrar estadísticas a intervalos de tiempo durante la ejecución, a fin de estudiar el comportamiento dinámico del sistema.
- *Lenguaje*: el lenguaje de programación debería ser capaz de describir el estado del modelo en cualquier momento. Esto significa que tanto entidades permanentes como temporales deben ser definidas de una manera simple. La

selección de componentes dependerá del lenguaje utilizado.

- *Control de la simulación*: para permitir el cambio dinámico de estado, deben estar disponibles algunos comandos para controlar componentes del modelo internos y externos.
- *Facilidad de operación*: este requerimiento incluye muchas características: verificación de errores y facilidades de *debugging*, legibilidad de los programas, inicialización de los modelos y pruebas de estabilidad, la habilidad para realizar sucesivas ejecuciones, facilidad de modificación ya sea durante o entre ejecuciones, posibilidad de construir modelos útiles desde un subconjunto básico de instrucciones y, finalmente, modularidad de las instrucciones y de la estructura del modelo.

A la hora de elegir un lenguaje para desarrollar un modelo de simulación, el analista puede utilizar un lenguaje de propósito general, como Java o C++, o un lenguaje específico para simulación. Los lenguajes de propósito general otorgan un mayor grado de flexibilidad, pero queda a cargo del analista desarrollar las características arriba listadas, lo que es propenso a errores, poco reusable, y posee un costo muy elevado. Una alternativa intermedia a esta aproximación es el uso de una extensión de algún lenguaje de simulación que provea primitivas para construir y manipular las entidades del sistema [Nance84][Okeefe86][Malloy86]. Esto le evita al analista tener que aprender un nuevo lenguaje de programación, así como también adquirir un software de simulación que, por lo general, suele ser costoso.

Los lenguajes específicos para simulación han evolucionado constantemente desde que surgieron, a principios de los años 60. La evolución ha estado guiada por numerosos factores, entre los que se destacan el incremento en la capacidad de cálculo y almacenamiento de las computadoras, el avance de los lenguajes de programación de propósito general, y los tipos de aproximaciones y técnicas para simular sistemas cada vez más complejos. Así, surgieron lenguajes para sistemas de colas [Anderson84][Kur86][Sinclair86], para aplicaciones de manufactura [Farnsworth86] [White86][Conway86][Steudel87][Ford87], sistemas basados en reglas y orientados a objetos, combinando conceptos de simulación e inteligencia artificial [Shannon87][Adelsberger86], sistemas de simulación de sistemas continuos

[Lee02][Mitchell76][Zeigler00], lenguajes para eventos discretos paralelos [Kleine71] [Vlatka00][Kachitvicyanukul90][Low99], entre otros. Dada la enorme variedad de variantes y alternativas existentes, esta tesis hace énfasis sólo en los lenguajes de simulación de eventos discretos, cuyas características principales se detallan a continuación.

Lenguajes de simulación de eventos discretos

Los lenguajes de simulación de eventos discretos pueden clasificarse de acuerdo a la *visión del mundo* que emplean [Hopper86]: planificación de eventos (*event scheduling*), búsqueda de actividades (*activity scanning*) e interacción de procesos (*process interaction*). Una visión del mundo es la perspectiva, o el punto de vista, desde el cual el modelador mira el mundo o el sistema a ser modelado. La visión del mundo elegida por el modelador influencia fuertemente la estructura y el método para construir el modelo (Imagen 17).

Todas las simulaciones contienen una *rutina de ejecución*, que es responsable de la secuenciación de eventos y de llevar adelante la simulación. En cada programa de simulación, la rutina de ejecución recupera el siguiente evento planificado, avanza el tiempo de simulación, y transfiere el control a la rutina de operación apropiada. Las rutinas de operación son dependientes de la visión del mundo utilizada, y pueden ser eventos, actividades o procesos. Estas operaciones constituyen los principales componentes de un modelo de simulación.

En la aproximación por planificación de eventos, un modelo o lenguaje de simulación contiene eventos y se preocupa por su efecto sobre el estado del sistema. Cada tipo de evento posee una rutina de evento correspondiente, que es llamada por la rutina de ejecución cuando un evento de ese tipo ocurre. El *calendario* contiene una lista de eventos incondicionales. Cualquier verificación de condición, que no sea del tiempo del reloj, se realiza dentro de las rutinas de eventos.

La búsqueda de actividades requiere que la rutina ejecutiva utilice tanto tiempo planificado como verificación de condiciones para seleccionar el siguiente evento del calendario. El componente básico del modelo es la actividad, la cual es un período de

tiempo que representa alguna operación en la que algunas entidades del sistema están comprometidas. Una actividad se define por dos eventos, uno que inicia el comienzo de la actividad y otro que marca su fin. Cada actividad contiene acciones y condiciones a verificar, y la actividad se ejecutará sólo cuando sus condiciones se satisfagan. La rutina de ejecución especifica qué es lo que se debe hacer para completar una actividad una vez que sus condiciones se hacen verdaderas. Para ello, busca todas las actividades elegibles de acuerdo al tiempo de planificación y en las cuales las condiciones se satisfacen, y ejecuta las acciones de la primera actividad encontrada. Cuando estas acciones son completadas, la rutina de ejecución comienza la búsqueda de una nueva actividad, repitiendo el proceso hasta que la simulación finaliza.

La visión del mundo basada en interacción de procesos combina algunas características de la planificación de eventos con otras de la búsqueda de actividades. En este caso, la simulación consta de un conjunto de procesos, o sea un conjunto de sentencias que describen operaciones en las cuales una entidad estará comprometida durante su tiempo de vida. Esta aproximación es la más intuitiva, en el sentido de que describe la historia de cada clase de entidad en el sistema.

Los modelos que utilizan interacción de procesos son muchas veces representados como un diagrama de flujo que muestra cómo la entidad se movería a través de sus procesos. Una entidad puede ser bloqueada o retrasada por algún motivo en varios puntos en el proceso. Un retraso incondicional ocurre hasta que cierta cantidad de tiempo ha transcurrido. Un retraso condicional es aquel en el que la duración depende del estado del sistema. Una entidad puede ser detenida en un punto de retraso hasta que es reactivada por la finalización de un retraso planificado, o porque una condición se ha hecho verdadera.

Si bien existen varias aproximaciones para implementar esta estrategia, la más común es la empleada en la implementación de GPSS, en la cual existe una lista de eventos futuros y otra lista de eventos actuales, y en la que los cambios de reloj provocan la búsqueda y movimiento de transacciones desde la lista de eventos futuros a la de eventos actuales [Zeigler84]. Este mecanismo será descrito con mayor detalle en la sección dedicada a GPSS.

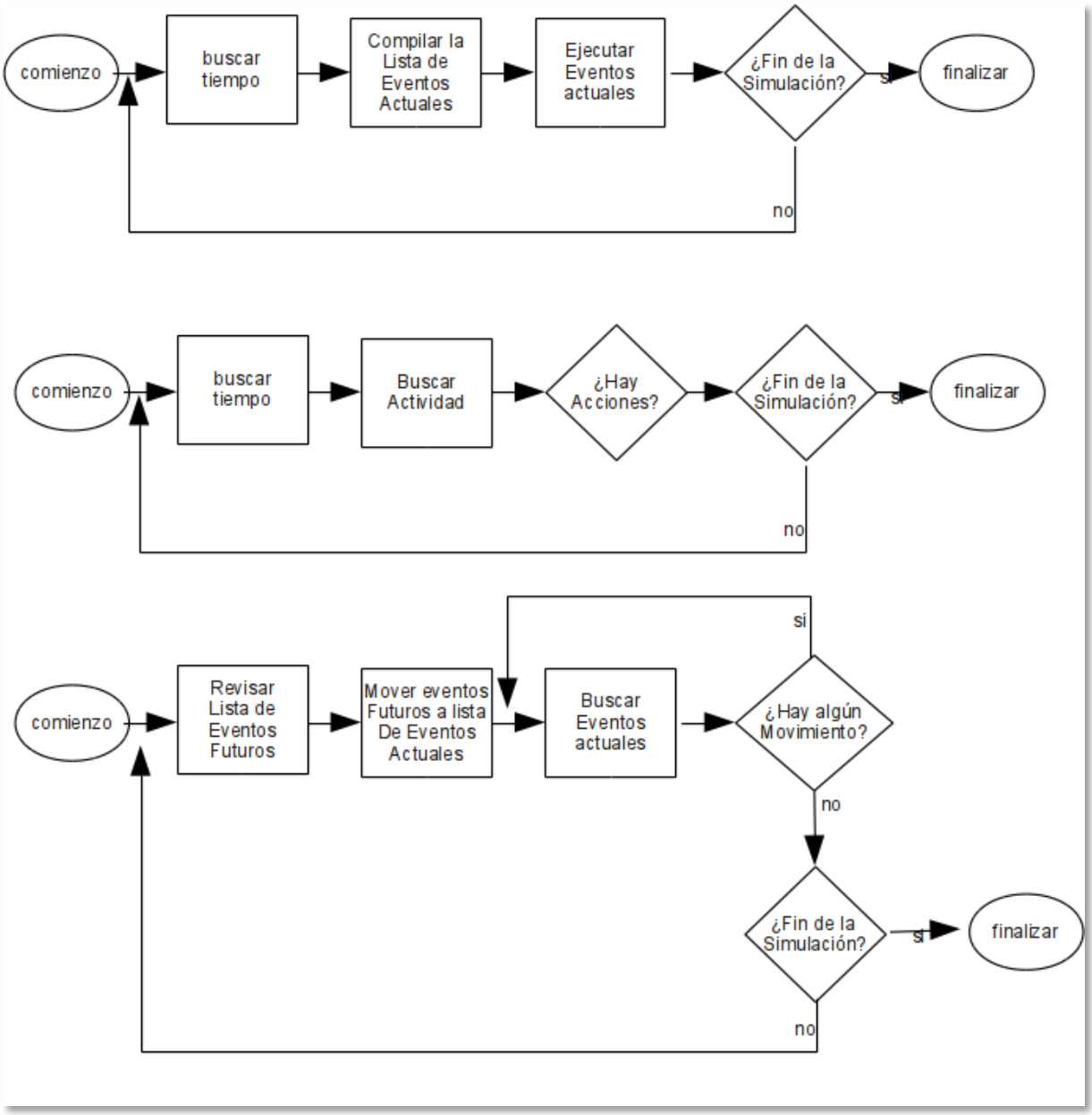


Imagen 17: Representación de las distintas visiones del mundo: planificación de eventos (arriba), búsqueda de actividades (centro) e interacción de procesos (abajo) [Pollacia]

Conceptos básicos de GPSS

El Sistema de Simulación de Propósito General, GPSS, nace en el año 1961 desarrollado por G. Gordon, quien trabajaba en ese momento en los laboratorios de IBM. A pesar de su antigüedad, es remarcable el intenso desarrollo que aún continúa sobre este lenguaje y la cantidad de nuevas versiones e innovaciones que se han generado en los últimos años: GPSS World for Windows (GPSSw) y WebGPSS (GPSS para la web)

[Stahl01] [Herscovitch65] [Crain97]. Esta continua evolución le ha permitido posicionarse desde hace varios años y mantenerse en la actualidad como una de las principales herramientas elegidas a la hora de enseñar en el área de Modelos y Simulación, no solo en ingeniería e informática, sino también en administración y negocios [McHaney96] [Herper99] [Banks82] [Herper03] [Sta00].

GPSS puede ser estudiado desde diferentes puntos de vista: como un lenguaje de programación, con sus sentencias, palabras claves y organización de código, como un motor de simulación, con sus entidades y su mecanismo de planificación y ejecución, y como herramienta de análisis, con sus funciones para conducir experimentos y los distintos tipos de reportes que genera.

El lenguaje de programación GPSS

Los modelos de simulación desarrollados en GPSS se componen de uno o más archivos (en casos de modelos muy complejos) que contienen la estructura del modelo y las definiciones de las entidades que se utilizarán durante la ejecución de la simulación. Esto se realiza a través de los dos únicos tipos de sentencias de GPSS: comandos y bloques. Los bloques son las instrucciones básicas de GPSS, y referencian entidades en sus operaciones. Estas referencias pueden realizarse mediante un número identificador de entidad, o a través de una etiqueta, que aporta mayor legibilidad al código.

En el nivel lógico, los elementos o entidades con sus atributos definen la estructura estática del modelo, la cual se completa con su estado en cualquier momento dado. Tales entidades son creadas cuando son requeridas por primera vez por el programa, a partir de una colección de entidades predefinidas disponibles para el programador. Al momento de compilar el programa, durante la fase de ensamble, una vez que se ha revisado el código en busca de errores de sintaxis y que se ha transformado el código fuente en una representación interna que permite su ejecución, se crean todas las entidades permanentes posibles. Sin embargo, algunas de estas entidades permanentes no pueden crearse al compilar el programa, pues su existencia dependerá de la secuencia de ejecución del programa. Por ejemplo:

```
1 SEIZE cajero
2 SEIZE RN1
```

El bloque SEIZE provoca que la transacción activa se adueñe de la entidad facilidad que referencia. En el ejemplo anterior, al ejecutarse la línea 1, la facilidad intentará tomar la facilidad "cajero". Si ésta estuviera libre, la transacción proseguirá con el siguiente bloque. De lo contrario, la transacción quedará a la espera de que la facilidad cajero se libere. En la línea 2, no se referencia una facilidad de manera explícita, sino que se utiliza un generador de números aleatorios (Random Number Generator, RN). Esa entidad generará un valor aleatorio, entre 000 y 999, el cual será el número de la entidad que la transacción activa intentará tomar por medio del bloque SEIZE. Este ejemplo simple, si bien de poca aplicación real, muestra un caso concreto en el cual el compilador no puede a priori crear todas las entidades permanentes. Sólo podrá crear la facilidad cajero en tiempo de compilación, pues es la única que se indica de manera explícita.

Las entidades temporales, o transacciones, son creadas cuando es requerido por la simulación durante la ejecución del programa, y son creadas por los bloques GENERATE y SPLIT. El movimiento de las transacciones a través de la lista de bloques es controlada por las operaciones de los bloques a los que acceden; estas operaciones definen el comportamiento dinámico del modelo.

Algunos atributos son definidos y actualizados de manera automática por el simulador GPSS, mientras que otros pueden ser especificados y gestionados por el programador. Los atributos estadísticos de las entidades permanentes son mantenidos de manera automática; sin embargo, GPSS también permite que el usuario recolecte sus propias estadísticas adicionales. Esta flexibilidad asegura que algunas estadísticas siempre estarán disponibles, y en caso de ser necesario, el programador podrá ampliarlas y obtener un mayor nivel de detalle.

Desde el punto de vista del nivel de implementación, un programa en GPSS consta de un conjunto bloques interconectados, que conforman la red de bloques y que constituyen las instrucciones básicas de GPSS. Cada uno de estos bloques posee una o más rutinas internas, y realiza una tarea concreta sobre una entidad del sistema, ya sea

permanente o temporal. La ejecución de las rutinas de los bloques es disparada por un evento de arribo de una transacción a dicho bloque, con la única excepción del bloque GENERATE. En este caso particular, GPSS debe crear una transacción antes que nada pueda moverse a través del modelo. La creación de esta primera transacción constituye el primer evento exógeno que debe planificarse y ejecutarse en el tiempo apropiado. Subsecuentemente, todos los bloques operan en la manera estándar ejecutando la función apropiada cuando son activados por una transacción. El movimiento de transacciones entre los bloques es controlado de manera automática por el simulador. El usuario no necesita a priori tener un profundo conocimiento de ese aspecto de la simulación, a menos que desee modificar el procesamiento estándar, requerimiento común cuando se está lidiando con sistemas muy complejos. Cuando una transacción ingresa a un bloque TERMINATE, es destruida y todas sus referencias a entidades del modelo son eliminadas.

El estado del modelo GPSS se modifica a medida que los eventos se suceden. A medida que las transacciones se mueven de un bloque al siguiente, los atributos de las entidades permanentes y temporales son modificados, lo cual puede involucrar cálculos complejos. La mayoría de estos cálculos son controlados directamente por GPSS durante la ejecución de las rutinas de los bloques.

Las entidades computacionales que posee GPSS son los generadores de números aleatorios (RN), vistos en el ejemplo anterior, las funciones (FUNCTION) y las variables (VARIABLE). Las variables son expresiones que se evalúan al momento de su invocación, y su resultado puede ser un valor numérico o un booleano. Las funciones pueden ser discretas o continuas, pueden tener argumentos numéricos o atributos de otras entidades, y proveen un método sencillo para incluir datos estadísticos y experimentales en el modelo. Mediante el uso de funciones de distribución acumuladas, el analista puede también especificar variables aleatorias.

Las transacciones poseen varios atributos internos propios, y el GPSS permite al programador almacenar valores adicionales en cada transacción, mediante el uso del bloque ASSIGN. Estos valores son internos a cada transacción y, por lo general, no son compartidos con otras transacciones del sistema. Por el otro lado, GPSS cuenta también con una entidad, llamada SAVEVALUE, que permite al programador almacenar valores

globales, accesibles a todas las transacciones. En los casos donde se requiera gestionar arreglos de datos, el GPSS posee una entidad llamada MATRIX, en la cual el usuario puede especificar una estructura de hasta dos dimensiones.

Entidades de GPSS

El GPSS posee un tipo de entidad temporal, llamada Transacción, y cuenta con un amplio conjunto de entidades permanentes, a partir de las cuales el programador puede crear instancias dentro de su modelo. Cada tipo de entidad permanente posee un propósito y un comportamiento diferente, recolecta distintos datos estadísticos, y se accede a algunas de sus funciones a través de uno o más bloques particulares. Algunos bloques sirven para alterar el estado interno de algunas entidades permanentes, mientras que otros sirven sólo para *consultar* el valor de algún atributo interno.

Para acceder al estado de las entidades, GPSS utiliza atributos numéricos estándares (Standard Numerical Attributes, SNA). Estos atributos son funciones que retornan el valor de un atributo específico de una entidad particular. Dicho de otro modo, los SNA permiten acceder a las variables de estado de la simulación. Estas variables de estado pueden almacenar números, incluyendo los valores booleanos (0 para falso, y 1 para verdadero), y también *strings* en algunos casos.

Se puede clasificar los SNA en dos categorías:

- *SNA tácitos*: son aquellos donde la entidad no requiere ser especificada, ya sea porque es única y global, o porque hacen referencia a la transacción activa. Algunos ejemplos de SNA de entidades globales son AC1, que retorna el valor absoluto del reloj del sistema, y TG1 retorna el valor del contador de terminación de la simulación (*termination count*). Entre los SNA que referencian a la transacción activa se encuentran el A1, que retorna el conjunto de ensamble de la transacción, el M1, que retorna su tiempo de tránsito, y el XN1, que retorna su identificador.
- *SNA explícitos*: aquellos que requieren que se indique la entidad sobre la cual evaluar algún atributo, ya que no se trata de una entidad global ni única. Si bien esto varía según el contexto de aplicación, en líneas generales estos SNA se

identifican por una clase, el símbolo $, y la entidad en cuestión. Algunos ejemplos:

- ✓ *W$cobrar*: retorna la cantidad de transacciones esperando ingresar al bloque con la etiqueta *cobrar*.
- ✓ *CH$cadena*: retorna el contenido actual de la Userchain llamada *cadena*.
- ✓ *CM$cadena*: retorna el contenido máximo de la Userchain llamada *cadena*.
- ✓ *F$cajero*: retorna 1 (verdadero) si la facilidad *cajero* se encuentra ocupada, o 0 en caso contrario.
- ✓ FI$cajero : retorna 1 (verdadero) si la facilidad *cajero* ha sido interrumpida, o 0 en caso contrario
- ✓ *CF$cajero*: retorna la cantidad de transacciones que han tomado la facilidad *cajero*.
- ✓ *X$stock*: retorna el valor del SAVEVALUE denominado *stock*.
- ✓ *P$costo*: retorna el contenido del parámetro *costo* de la transacción activa o de la transacción que está siendo evaluada con ciertos bloques como EXAMINE, SCAN y ALTER.

La lista de entidades de GPSS es extensa [Minuteman13], y continúa expandiéndose a medida que las nuevas implementaciones incorporan nuevas funciones y herramientas avanzadas. Este texto no busca abarcar la totalidad de las entidades, sino que se centrará en las principales desde el punto de vista de su aplicación en la enseñanza, las mayoría de las cuales existen desde las primeras versiones de GPSS. Todas las entidades que aquí se detallan están incluidas como temas básicos del curso de Modelos y Simulación, y la mayoría de ellas fueron también implementadas en la herramienta GPSS Interactivo, descrita con mayor detalle en el capítulo 4. El listado siguiente contiene las entidades, junto a una breve descripción de su propósito y funcionamiento, y los bloques que se utilizan para accederlas y modificarlas.

Transaction (Transacción)

Las transacciones son las entidades temporales de GPSS, también conocidas como unidades de tráfico. Son creadas por el bloque GENERATE y destruidas por el bloque

TERMINATE. Luego que es generada, una transacción ingresa al siguiente bloque secuencial (Next Sequential Block, NSB) y a partir de allí se mueve por la red de bloques, en tiempo de simulación cero, hasta que sea detenida por alguna de las siguientes condiciones:

- Un tiempo de retraso impuesto por el bloque ADVANCE (que puede representar, por ejemplo, un tiempo de servicio).
- El acceso al siguiente bloque secuencial fue denegado (por ejemplo, porque una condición no se cumplió).
- La transacción fue desactivada (ingresó a una cadena de usuario).
- Destrucción de la transacción (bloque TERMINATE).

Las transacciones pueden dividirse (bloque SPLIT) para formar un conjunto de ensamble (*assembly set*), y ser luego reunidas (bloque GATHER), combinadas (bloque ASSEMBLE) o coordinadas con otras (bloque MATCH). Estas características resultan muy útiles para implementar mecanismos de sincronización y/o división de tareas, como por ejemplo un sistema en el cual un proceso se divide en un conjunto de sub-operaciones simultáneas que deben ser realizadas en fases.

Cuando una transacción se mueve a través del diagrama de bloques, altera el estado del sistema. A este cambio de estado se lo denomina **evento.** Por lo tanto, los eventos son implícitamente partes del funcionamiento dinámico de las transacciones. Para controlar la secuencia lógica de la simulación, GPSS organiza las transacciones en cadenas (*chains*), que son en realidad listas ordenadas según algún criterio. Como se verá a continuación, el criterio varía según el propósito de la cadena. Existen varios tipos de cadenas disponibles para modelar una gran variedad de situaciones; sin embargo, dos de ellas, que son definidas y mantenidas automáticamente por el motor de simulación, se encuentran en cualquier programa escrito con GPSS:

1. La cadena de eventos actuales (Current Events Chain, CEC).
2. La cadena de eventos futuros (Future Events Chain, FEC).

La CEC contiene todas las transacciones que deben moverse en la unidad de tiempo actual, o sea, antes de avanzar el reloj de simulación. Estas transacciones están listas

para avanzar dado que las condiciones que las habían frenado ya han sido removidas. La FEC contiene transacciones que deberán avanzar en algún tiempo de simulación futuro. Se utiliza el atributo de la transacción llamado "tiempo de partida de bloque" (Block Departure Time, BDT) para representar el momento en el cual la transacción debe ser planificada para avanzar, siempre y cuando no existan condiciones lógicas que la detengan. Entonces, siendo C1 el tiempo actual del reloj, se cumple que:

1. La CEC posee transacciones con BDT ≤ C1
2. La FEC posee transacciones con BDT > C1

La definición de los eventos requiere, además del BDT, otros atributos propios de las transacciones, como el bloque actual (Current Block Location, CBL), el NSB (Next Sequential Block) y la prioridad (Priority, PR).

Durante la simulación, los eventos deben ejecutarse cronológicamente. Para asegurar esto, las transacciones en las cadenas son mantenidas en orden ascendente a partir de su BDT. Esto sucede exactamente así en la FEC; si más de una transacción ingresa a dicha cadena con el mismo BDT, se organizarán según el orden de llegada (FIFO). La CEC realiza una tarea adicional cuando las transacciones tienen distintas prioridades. Las transacciones de más alta prioridad son puestas delante de las de menor prioridad; nuevamente, a igual prioridad, se considera el atributo BDT, y a igual prioridad e igual BDT, se tomará el orden de llegada.

Para ejecutar eventos, se analiza la CEC para determinar si una transacción puede avanzar. En tal caso, se mueve tanto como es posible, hasta que es detenida por alguna condición, como se vio previamente. Si una transacción no puede avanzar, continúa el análisis de la CEC. Cuando una transacción se mueve, puede alterar el estado del modelo y permitir continuar a transacciones que habían sido previamente detenidas. Por lo tanto, cuando el estado del modelo es alterado, el motor de simulación de GPSS debe volver a analizar la CEC desde el principio. Este proceso continúa, sin modificar el tiempo de reloj actual, hasta que o bien la CEC se ha vaciado, o ningún miembro de la misma puede moverse. En este momento, el reloj de simulación se actualiza con el valor del siguiente evento (menor BDT de la FEC), y todas las transacciones con ese BDR son transferidas de la FEC a la CEC.

Facility (Facilidad)

La facilidad es una de las entidades más importantes en GPSS. Se utiliza, en la mayoría de los casos, para representar servidores capaces de atender a un cliente a la vez, e implementan varios sistemas de esperas para aquellos clientes que desean acceder al servicio pero no pueden hacerlo porque otro cliente está haciendo uso del servidor en ese momento.

Una facilidad es una entidad que posee varios atributos, entre los cuales el más importante es su propietario (*ownership*). Una facilidad puede ser tomada por una única transacción, en cuyo caso se dice que está ocupada (*busy*). Si ninguna transacción la ha tomado, se dice que la facilidad está libre (*idle*). Una facilidad no puede ser liberada por una transacción que no la había tomado (esto no sucede así con los almacenes). Para tomar una facilidad, las transacciones deben ingresar satisfactoriamente a un bloque SEIZE o a un bloque PREEMPT. El bloque PREEMPT se utiliza para desplazar una transacción que había tomado una facilidad (por ejemplo, para representar un proceso de mayor prioridad). Si una transacción no puede tomar una facilidad, se coloca en una de las cadenas de la facilidad.

Las facilidades poseen varias líneas o cadenas, para aquellas transacciones que esperan que algún evento relacionado con la facilidad suceda (Imagen 18). Cada facilidad tiene una cadena de retraso (*delay chain*), que se utiliza para las transacciones que están esperando que se libere la facilidad, una cadena de pendientes (*pending chain*) para transacciones que intentaron desplazar a otra transacción de la facilidad (mediante un bloque PREEMPT), pero que no pudieron porque la preempción se intentó realizar en *modo interrupción* y la transacción que posee actualmente a la facilidad lo hizo también mediante una preempción, y una cadena de interrupción (*interrupt chain*) para las transacciones que fueron desplazadas de la facilidad pero aún deben volver a tomarla. Las transacciones que se encuentran en la cadena de retraso, cadena de interrupción o la cadena de pendientes de una facilidad se dice que compiten por dicha facilidad. En algún momento, cada transacción que toma la facilidad deberá liberarla, mediante los bloques RELEASE o RETURN, con lo cual alguna de las transacción que están compitiendo podrá acceder a la facilidad.

Las transacciones que no pudieron ingresar al bloque SEIZE, son puestas en la cadena

de retraso de la facilidad, ordenadas por prioridad primero y FIFO después. Cuando la facilidad se libera, la siguiente transacción dueña se elige entre los ocupantes de las cadenas de transacciones de la facilidad. Primero se buscan transacciones con preempciones pendientes, de la cadena de pendientes, luego transacciones que fueron desplazadas de la facilidad, de la cadena de interrupción, y finalmente transacciones esperando normalmente por la facilidad, de la cadena de retraso.

Las facilidades pueden estar libres (*idle*) y ocupadas (*busy*), como consecuencia de la posesión y liberación por parte de las transacciones, y también pueden estar disponibles (*available*), en cuyo caso se dice que "funcionan" con normalidad o deshabilitadas (*unavailable*), en cuyo caso se le niega el acceso a cualquier transacción, sin importar si lo hace mediante un bloque SEIZE o un bloque PREEMPT. Se utilizan los bloques FAVAIL y FUNAVAIL para cambiar el estado entre disponible y deshabilitada respectivamente.

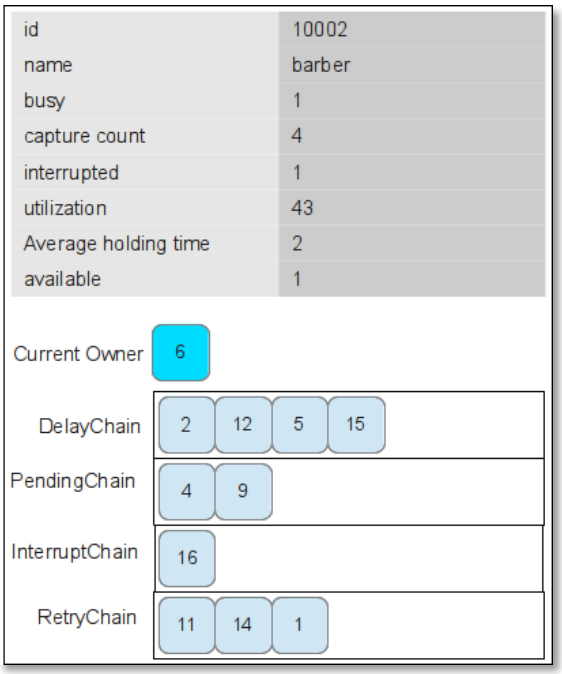


Imagen 18: Atributos y cadenas de una facilidad

Storage (Almacén)

Una entidad almacén se asocia con un número de unidades de almacenamiento que son ocupadas y liberadas por transacciones. Los almacenes pueden ser utilizados como conjuntos de recursos disponibles, y permiten controlar el flujo de transacciones en el

modelo.

Cuando una transacción ingresa a una entidad almacén, mediante un bloque ENTER, utiliza u ocupa una o más unidades del almacén. Si la cantidad de unidades que la transacción requiere ocupar (especificada en el bloque ENTER) no puede satisfacerse, se le denegará el acceso al bloque y será enviada a la cadena de retraso del almacén, donde deberá esperar hasta que otras transacciones liberen el suficiente espacio (mediante el bloque LEAVE) para que pueda ingresar al bloque ENTER.

La capacidad del almacén puede ser liberada por cualquier transacción, incluso si no había ingresado antes a un bloque ENTER de dicho almacén. Sin embargo, si la transacción intenta liberar más capacidad de la que se declaró en el almacén, surgirá una condición de error que detendrá la simulación.

Cuando una transacción ingresa al bloque LEAVE y libera una o más unidades del almacén, se buscan otras transacciones de la cadena de retraso del almacén, que puedan satisfacer su demanda de unidades. Se utiliza la disciplina de "la primera que encaja", lo cual significa que se analiza cada transacción de la cadena de retraso para ver si puede ejecutar el bloque ENTER con la capacidad que demanda, comenzando por la de más alta prioridad. Si se encuentra una, la transacción es removida de la cadena de retraso, se le permite ingresar al bloque ENTER e ingresa al final de la CEC. A continuación, continúa la búsqueda de transacciones en la cadena de retraso del almacén, hasta que todas las transacciones hayan sido analizadas, o hasta que la capacidad del almacén haya sido completada.

A diferencia de las facilidades, los almacenes deben definirse mediante el comando STORAGE, con lo cual se les asigna un nombre y una capacidad máxima. Los almacenes también pueden cambiar de estado entre vacío, lleno, y ni vacío ni lleno, y al igual que las facilidades, pueden estar disponibles o deshabilitados mediante los bloques SAVAIL y SUNAVAIL respectivamente.

Queue (Colas)

Las colas son entidades utilizadas para recolectar estadísticas, y pueden aceptar cualquier número de transacciones. Se utilizan para registrar el tiempo que una

transacción espera por un servicio o por una condición, y acumulan una gran cantidad de datos y estadísticas: contenido actual, número de entradas totales, número de entradas totales sin esperar, contenido máximo en algún momento, relación cantidad-tiempo, entre otros. Se utilizan los bloques QUEUE (encolar) y DEPART (desencolar) para actualizar las estadísticas de una cola; por lo general, estos bloques se colocan alrededor (por encima y por debajo) de otros bloques que podrían provocar una detención o espera de las transacciones, como SEIZE, PREEMPT y ENTER. Esta entidad mantiene las estadísticas de manera automática, y las incorpora en los reportes una vez finalizada la simulación.

Es importante aclarar que esta entidad representa la espera de transacciones en una cola, pero no a la cola en sí misma. Y esta entidad tampoco tiene una capacidad máxima (se supone ilimitada). Esto significa que en ningún caso se le niega a una transacción el ingreso a un bloque QUEUE o DEPART, y que la ubicación de estos bloques es crucial para determinar qué estadísticas se recolectarán. Por ejemplo:

```
1 GENERATE 5,3
2 QUEUE general
3 QUEUE esperaFacilidad
4 SEIZE facilidad
5 DEPART esperaFacilidad
6 ADVANCE 7,2
7 QUEUE esperaAlmacén
8 ENTER almacén,2
9 DEPART esperaAlmacén
10 ADVANCE 3,1
11 LEAVE almacén,2
12 DEPART general
13 TERMINATE
```

El ejemplo anterior utiliza tres colas (general, esperaFacilidad, esperaAlmacén). La cola "general" toma estadísticas desde que la transacción es generada (línea 1) hasta que sale del sistema (línea 13). La cola "esperaAlmacén" sólo recolecta estadísticas del tiempo de espera por la entidad "almacén", o sea para ingresar al bloque ENTER (línea 8). Por último, la cola esperaFacilidad recolecta estadísticas relativas a la espera para ingresar al bloque SEIZE (línea 4), o sea, la espera para tomar la facilidad.

Savevalue (Almacén de valor)

Una entidad savevalue se asocia con una variable que puede tomar cualquier valor. Este valor puede ser asignado o modificado por el bloque homónimo SAVEVALUE, y su valor puede consultarse mediante la función X seguido del nombre del savevalue.

Al momento de su creación, los savevalues toman el valor 0, aunque este valor inicial puede modificarse por medio del comando INITIAL.

Los savevalues funcionan de manera similar a los atributos de las transacciones, ya que pueden contener cualquier valor, pero a diferencia de estos, cada savevalue puede ser accedido por cualquier transacción. Es común realizar una analogía entre esta entidad y una variable global en un lenguaje de programación procedural. Por lo general, los savevalues se utilizan como contadores globales, o para especificar condiciones de parada para todas las transacciones de la simulación.

Userchain (Cadena de usuario)

Las cadenas de usuario son entidades que pueden aceptar cualquier número de transacciones, las cuales son desactivadas cuando ingresan a la cadena. Esta es una característica muy poderosa de GPSS, y posee muchos usos. Pueden servir para acelerar la ejecución de un programa, gestionando las transacciones retrasadas por fuera del motor de GPSS y evitando que sean procesadas hasta que sea el momento indicado (esto se verá con mayor detalle más adelante), y también pueden servir para generar mecanismos de encolamiento específicos, a partir, por ejemplo, de un atributo de las transacciones.

Las cadenas de usuario pueden manipularse mediante los bloques LINK y UNLINK, y

poseen un atributo particular llamado indicador de enganche, o "link indicator", que sirve para conocer si la cadena posee alguna transacción al momento de ejecutar un bloque LINK.

Function (Función)

Las funciones de GPSS se utilizan para calcular y retornar un valor derivado de un argumento o parámetro de entrada, que puede ser un número aleatorio o cualquier atributo numérico[8] de cualquier entidad del modelo. Una función se define con el comando FUNCTION, una sentencia que especifica su argumento, y un operando que indica tanto el tipo de la función como el número de pares de datos que se aparean con las sentencias que conforman el cuerpo de la función.

Existen 5 tipos de funciones:

1. funciones continuas (tipo C), que realizan una interpolación linear;
2. funciones discretas (tipo D), en las cuales a cada argumento se le asigna un valor separado;
3. funciones de atributo evaluado (tipo E), donde cada valor del argumento se asigna a un atributo numérico (SNA) que será evaluado;
4. funciones de lista (tipo L), que utilizan el argumento para determinar la posición del valor a retornar de una lista de valores;
5. funciones de lista evaluada (tipo M), que utilizan el argumento para determinar la posición del atributo numérico a evaluar de una lista de atributos numéricos (SNA)

Variable (Variable)

Una entidad variable representa una expresión compleja que puede ser calculada bajo demanda. Todas las entidades variables pueden definirse como expresiones GPSS, y pueden incluir constantes, funciones aritméticas de la librería de GPSS, invocaciones a atributos de otras entidades mediante SNA, y operadores aritméticos y lógicos. Las

[8] Los atributos numéricos de GPSS, conocidos como SNA (Standard Numerical Attributes) son funciones que permiten acceder a los atributos internos de las entidades GPSS.

variables se definen con alguno de los comandos VARIABLE (variable entera), FVARIABLE (variable en coma flotante), o BVARIABLE (variable booleana, que retorna 1 para verdadero, 0 para falso).

Group (Grupo)

La entidad grupo representa un conjunto, que puede contener o bien transacciones, o bien números. Los grupos numéricos suelen utilizarse para registrar eventos o para describir el estado de un proceso que está siendo simulado. Los grupos de transacciones son útiles para clasificar transacciones según diversos criterios (una transacción puede pertenecer a más de un grupo, y un grupo puede tener un número ilimitado de transacciones). Las transacciones ingresan a un grupo utilizando el bloque JOIN, y salen del grupo con el comando REMOVE.

Una característica interesante de los grupos de transacciones es que GPSS posee bloques para buscar si alguna transacción de un grupo cumple con cierta condición, y para modificar atributos de todas las transacciones de un grupo que satisfacen algún criterio (bloques EXAMINE, ALTER y SCAN).

Assembly Set (Conjunto de Ensamble)

Todas las transacciones poseen un atributo interno, denominado conjunto de ensamble (Assembly set, AS). El AS brinda un mecanismo para asociar varias transacciones, y permite implementar mecanismos para sincronizar la ejecución de grandes conjuntos de transacciones.

Al momento de su nacimiento, mediante un bloque GENERATE, cada transacción pertenece a un conjunto de ensamble propio, identificado con el mismo número de la transacción. Sin embargo, si el nacimiento de una transacción se realiza por medio del bloque SPLIT, la nueva transacción tomará el mismo conjunto de ensamble que el del la transacción activa (aquella que ejecutó el SPLIT). Además, las transacciones pueden cambiar su conjunto de ensamble mediante el bloque ADOPT.

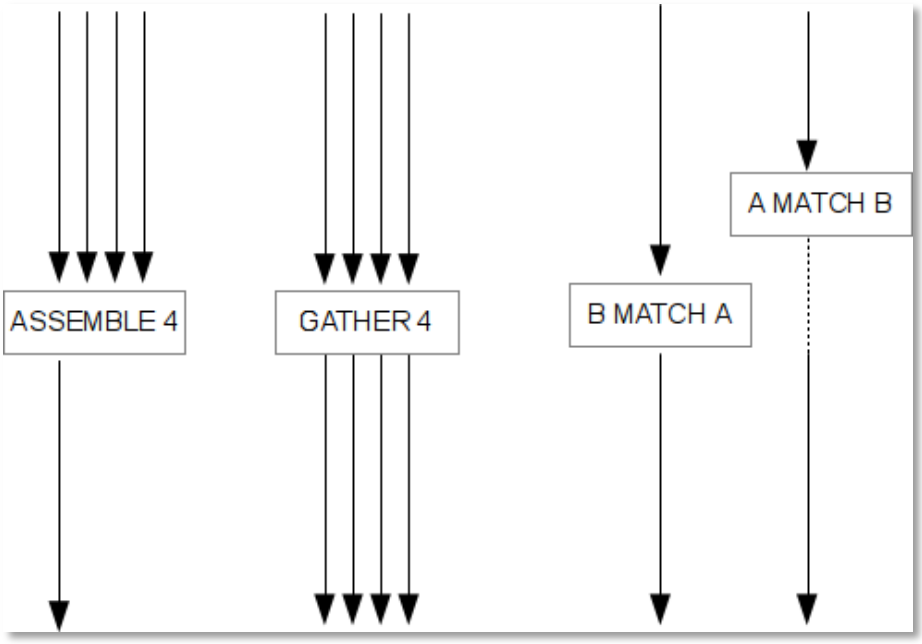


Imagen 19: Sincronización de transacciones mediante ASSEMBLE, GATHER y MATCH

La sincronización de transacción se realiza mediante bloques GATHER, ASSEMBLE y MATCH (Imagen 19). Estos tres bloques operan siempre sobre transacciones del mismo conjunto de ensamble. El bloque GATHER X (donde X es un número o una expresión numérica) detiene a todas las transacciones hasta que X transacciones del mismo conjunto de ensamble arriben. Cuando esto sucede, estas X transacciones son liberadas y pueden continuar su ejecución. El bloque ASSEMBLE X (nuevamente, X debe ser una expresión numérica) realiza la misma operación que GATHER, pero cuando las X transacciones han arribado, sólo la primera puede continuar y las restantes son destruidas. Por último, el bloque MATCH permite a dos transacciones sincronizar su ejecución en dos lugares de la red de bloques. Cuando una transacción arriba a un bloque MATCH, espera allí hasta que otra transacción con su mismo conjunto de ensamble arribe a otro bloque MATCH. Cuando esto sucede, ambas transacciones pueden seguir su ejecución.

Block (Bloque)

Los bloques son las sentencias básicas de GPSS, y a la vez son también entidades del modelo. Los bloques recolectan estadísticas básicas sobre la cantidad de transacciones que han ingresado y las transacciones que actualmente se encuentran en cada bloque.

La secuencia de bloques por la que pasan las transacciones determina la naturaleza y buena parte del resultado de cualquier simulación. Cada transacción ingresa a un bloque, y luego al siguiente, hasta que ingresa a un bloque TERMINATE (o hasta que la simulación finaliza). Muchos bloques requieren que se cumplan ciertas condiciones para que la transacción pueda ingresar (facilidad libre, almacén con espacio suficiente, etcétera). A cada tipo de bloque se le asocia una acción que afectará a otras entidades de la simulación, todo bajo la supervisión del planificador de transacciones. En general, cada bloque determina si la transacción activa puede ingresar, en cuyo caso se actualizarán estadísticas del bloque, de la transacción y del sistema, y en la mayoría de los casos se planifica la ejecución del NSB para la dicha transacción.

Los bloques pueden poseer una etiqueta (*label*), que luego puede ser utilizada por bloques que ejecutan saltos o evalúan condiciones, como por ejemplo TEST, GATE y TRANSFER, o para enviar transacciones bajo ciertas situaciones como por ejemplo al ejecutar un bloque UNLINK, PREEMPT o REMOVE.

Ejecución de un programa GPSS: el planificador de transacciones

GPSS sigue, como se explicó más arriba, el modelo de interacción de procesos. Este modelo mantiene en listas separadas los eventos futuros y los eventos actuales, que en términos de GPSS se traduce en cadenas de transacciones. Antes de comenzar la simulación, el motor de simulación analiza el código ingresado por el usuario, y separa bloques de comandos. En esta etapa también realiza las comprobaciones sintácticas y semánticas correspondientes. Si los análisis fueron superados satisfactoriamente, los comandos son dispuestos en una cola de ejecución, y los bloques son organizados en una lista, según el orden de aparición en el código. Aquí también se identifican todos los bloques GENERATE, y para cada uno se planifica una transacción inicial que ingresará a su bloque GENERATE al comenzar la simulación. Luego, comienza la ejecución de la cola de comandos, los cuales consisten por lo general en declaraciones de funciones, variables y almacenes, inicializaciones de valores, y finalmente el comando START, el cual inicializa el contador de terminación y provocará la ejecución de las transacciones ya creadas. Estas transacciones ingresarán a sus bloques GENERATE, con lo cual se calcularán los parámetros que indican los tiempos de generación de transacciones. Con

este cálculo finalizado, se generará una nueva transacción y se planificará a futuro (o sea, se colocará en la FEC) con BDT igual al tiempo actual de reloj (cero al principio) más el valor calculado. Además, esta nueva transacción tendrá como NSB el mismo bloque GENERATE, lo que provocará la repetición de la secuencia de generación de transacciones hasta que finalice la simulación o hasta que se alcance el límite máximo indicado en el parámetro C del bloque GENERATE.

La rutina de ejecución de la simulación mantiene un reloj como la principal entidad que controlará la simulación, y escanea constantemente las cadenas de eventos actuales y de eventos futuros. Estos escaneos se relacionan con las operaciones de varios bloques, y por eso es importante tener en claro en qué consiste la función de escaneo. GPSS realiza la operación de escaneo en 3 fases:

1. Actualizar el reloj al siguiente tiempo de partida de bloque (BDT) más inminente, y transferir todas las transacciones con ese BDT desde la FEC hacia CEC. La organización de la FEC en orden ascendente del tiempo para el cual las transacciones fueron planificadas para dejar la cadena, asegura que las transacciones más inminentes se encontrarán al frente de la cadena.

2. Buscar en la CEC la primera transacción activa posible: la rutina de ejecución de simulaciones debe estar al tanto de todo lo que sucede en el sistema. Esto incluye mantener un registro del estado de cada entidad, monitorizar los cambios de estados, y transportar sus efectos sobre cada elemento concernido. Para realizar esto, se mantienen dos tipos de indicadores de estado, que cumplen un rol fundamental a la hora de escanear la CEC. El primer indicador, conocido como marca de cambio de estado (*status-change flag*), se utiliza para determinar la posición del escaneo, ya sea al principio de la CEC o en la siguiente transacción secuencial. El segundo indicador se encuentra en cada transacción, y se llama indicador de estado de escaneo (*scan-status indicator*). Si este indicador está activado (*on*), la transacción se encuentra inactiva o retrasada, y por lo tanto será ignorada durante el escaneo en la CEC. Por el contrario, si el indicador está desactivado (*off*), la transacción estará en un estado de *potencialmente activa,* lo que significa que cuando la rutina de ejecución alcance esta transacción, intentará moverla hacia su NSB.

3. Intentar avanzar dicha transacción hacia su NSB: cuando una transacción se

encuentra como potencialmente activa (su indicador de escaneo está desactivado), la rutina de ejecución de transacciones intentará moverla. Aquí pueden surgir dos posibilidades: a) que la transacción no pueda ingresar al siguiente bloque, y b) que la transacción sí pueda ingresar al siguiente bloque.

a) Si la transacción no puede ingresar al siguiente bloque, puede deberse a que no ha podido acceder a la entidad que ha intentado obtener, por ejemplo una facilidad no disponible o un almacén completo. Aquí se debe activar el indicador de escaneo, para indicar que la transacción está inactiva y su estado actual es *retrasada*. Además, esta transacción es puesta en la cadena de retraso de la entidad en cuestión, que es simplemente una lista de todas las transacciones que no pueden avanzar hasta que el estado de la entidad que causó la desactivación de la transacción cambie. Otro motivo por el cual la transacción no puede avanzar es porque el NSB se lo ha negado. Esto sucede con bloques que evalúan condiciones, como TEST, GATE y TRANSFER (modos BOTH y ALL). En este caso, no se activa el indicador de escaneo de la transacción, y por lo tanto no se asocia a una *delay chain*, sino que se asocia a una *retry chain* (cadena de retraso), disponible en todas las entidades de GPSS. A continuación se tratará esto con mayor detalle.

b) Si la transacción sí puede ingresar al NSB, dependerá del bloque que se está ejecutando:

   I. Un bloque BUFFER o un bloque PRIORITY con la opción BUFFER: la transacción deja de estar activa (se envía al final de la CEC), y se escanea la CEC en busca de otra transacción.

   II. Un bloque TERMINATE, o un bloque ASSEMBLE donde un miembro del mismo grupo de ensamble ya está esperando. La transacción es eliminada del sistema.

   III. Un bloque LINK. La transacción se envía a una cadena de usuario, siempre y cuando el indicador de enlace (*link indicator*) esté activado.

   IV. Un bloque ASSEMBLE, con la transacción siendo la primera en entrar, un bloque GATHER o un bloque MATCH (y la transacción no encuentra su

conjugada en otro bloque MATCH). La transacción se marca como bajo una condición de *matching* interrumpido.

La marca de cambio de estado puede ser encendida o apagada por varios tipos de bloques. Luego que finalizó de mover la transacción activa, GPSS verifica el estado de esta marca de cambio de estado. Si la misma se encuentra activa, entonces la CEC es escaneada nuevamente desde el principio. Si se encuentra inactiva, entonces el escaneo continúa con la siguiente transacción secuencial. Existen esencialmente cuatro tipos de condiciones que pueden activar la marca de cambio de estado:

1. La marca se activa incondicionalmente por los bloques que cambian el estado de facilidades, almacenes y llaves lógicas. Esto asegura que todas las transacciones que están detenidas por los bloques SEIZE, PREEMPT, ENTER y GATE serán movidas si algún cambio en el estado de la entidad remueve la condición de bloqueo.

2. La marca se activa cuando se completa una operación de ensamble, reunión o *matching*, en los bloques ASSEMBLY, GATHER y MATCH.

3. Mediante un bloque PRIORITY, pero si posee la opción BUFFER, GPSS retomará la CEC desde el principio, y por lo tanto reiniciará la marca de cambio de estado.

4. Una transacción es removida de una cadena de usuario mediante un bloque UNLINK.

Gestión de transacciones en la Retry Chain

En la sección anterior se explicó el funcionamiento interno del planificador de transacciones de GPSS. Allí se expuso que, bajo ciertas condiciones, las transacciones pueden ser enviadas hacia una cadena de reintentos o *retry chain*, y que todas las entidades poseen una cadena este tipo. La cadena de reintentos contiene a todas las transacciones que, por algún motivo, fueron detenidas porque alguna condición no se satisfizo, condición que depende de algún atributo propio de la entidad. Esto sucede por lo general al utilizar los bloques TEST y GATE, y bajo ciertas condiciones en el bloque TRANSFER.

Las transacciones en las cadenas de reintentos permanecen desactivadas de la

ejecución de la simulación, pero son reactivadas cada vez que el estado de la entidad por la cual fueron detenidas es alterado por alguna transacción. Esto funciona de manera similar al patrón de diseño Observer [Gamma94] y, como se verá en el capítulo 5, esta visión es muy natural para los alumnos que empiezan a descubrir el lenguaje GPSS.

Al detectarse un cambio en el estado de una entidad *observada*, todas las transacciones que estaban desactivadas en su cadena de reintentos son reactivadas. Esto significa que el motor de simulación de GPSS realiza un movimiento masivo de transacciones desde una cadena de reintentos hacia la CEC, y da a todas las transacciones la posibilidad de continuar su ejecución, o lo que es lo mismo, de volver a evaluar la condición por la cual habían sido enviadas a la cadena de reintentos. Un problema particular que presenta este diseño se da cuando una de las transacciones vuelve a alterar el estado de la entidad en cuestión, de modo tal que hace que la condición que el resto de la transacciones debe evaluar se vuelva falsa. En este caso, todas las transacciones restantes que deben evaluar la condición volverán nuevamente a la cadena de reintentos de la entidad. Esto genera un movimiento innecesario de transacciones, lo cual puede degradar considerablemente la performance general del programa. Sin embargo, puede implementarse una solución muy simple a partir del uso de cadenas de usuario, a fin de evitar el encadenamiento automático de transacciones de GPSS, y manejarlo manualmente desde el código del programa. Como se verá más adelante, esta característica presenta muchas dificultades a los alumnos, pues para comprender el problema deben conocer profundamente la rutina de ejecución de transacciones.

Capítulo 4

GPSS Interactivo

Introducción

GPSS Interactivo (GPSSi) es un software cuyo diseño está orientado específicamente para la enseñanza del simulador GPSS; esto incluye tanto los aspectos propios del lenguaje de programación GPSS, su sintaxis y su semántica, como el funcionamiento interno del motor de simulación orientado a eventos y gobernado por colas. Si bien esta herramienta posee ciertas características pedagógicas, es preciso tener en cuenta que para su correcto aprovechamiento debe considerarse como una parte más en un curso de Simulación. Dicho de otro modo, la herramienta por sí sola no ayudará necesariamente a los alumnos a trabajar de manera apropiada con este simulador; por el contrario, el rol del docente y la organización del curso siguen siendo factores fundamentales al momento de la enseñanza, y esta herramienta es un soporte adicional. GPSSi es un medio para apoyar la enseñanza, y no un fin en sí mismo.

Es evidente que GPSSi no busca reemplazar ni mucho menos competir con otras herramientas de simulación profesionales, ni tampoco con otros entornos de desarrollo integrados (IDE) para GPSS. Estas herramientas fueron pensadas para simular modelos complejos y poseen herramientas de experimentación y análisis de datos muy diversas y avanzadas. Por el contrario, GPSSi sirve para ejecutar modelos simples, pues el objetivo es ayudar a los alumnos a comprender cómo estos modelos se han ejecutado y por qué se han alcanzado ciertos resultados. Una vez que los alumnos logran comprender esto, el rol del docente será ayudarlos a plasmar estos conceptos en simuladores más potentes, realizando una migración paulatina hacia estos entornos profesionales. Al fin y al cabo, estos simuladores son los que los futuros profesionales utilizarán una vez que su formación académica haya finalizado y, por lo tanto, es preciso que eventualmente trabajen con ellos.

En este capítulo se describen las principales características de la herramienta GPSS Interactivo. Entre estas características se incluyen aspectos del desarrollo, arquitectura y tecnologías, así como también las cuestiones de diseño orientadas a la enseñanza de GPSS.

Arquitectura de la aplicación

GPSS Interactivo fue diseñado para cumplir dos objetivos específicos en el marco de un curso de Modelado y Simulación:

- asistir al alumno durante la creación de modelos en lenguaje GPSS;
- ayudar al alumno a comprender cómo sus modelos son ejecutados.

El primer objetivo implica no sólo escribir código de programación en lenguaje GPSS, sino también comprender cómo se organizan las distintas porciones de código, cómo interactúan las sentencias con las entidades que GPSS define, y finalmente qué alternativas existen al momento de trabajar con una entidad, un bloque o un comando. Para lograr estos objetivos, GPSS Interactivo posee una interfaz de trabajo simple y fácil de entender, que permite al alumno trabajar con modelos en GPSS de manera interactiva —esto es, en base a diálogos y ayudas contextuales—. El segundo objetivo indica que esta herramienta debe ser capaz de ejecutar modelos en GPSS, lo cual significa interpretar código, generar modelos y entidades, y finalmente, simularlos respetando el mecanismo de ejecución del motor de simulación de GPSS.

Existe desde luego cierta dependencia entre los objetivos arriba mencionados, dado que al fin y al cabo ambos dependen del lenguaje GPSS, de su sintaxis, sus entidades y su simulador interno. Sin embargo, es evidente que hay una clara distancia entre el objetivo de asistencia en la creación de modelos válidos, y el de ayuda en la interpretación de resultados de simulación. Es por ello que GPSS Interactivo ha sido diseñado como dos aplicaciones diferentes, que pueden funcionar independientemente una de la otra, pero que también pueden integrarse para trabajar juntas como una única aplicación. Estas aplicaciones son el Entorno Interactivo para la creación de modelos, y el Motor de Simulación de GPSS con Persistencia (GPSS PSE). Para que ambas aplicaciones puedan comunicarse, intercambiar información y funcionar como una única aplicación integral

para los usuarios, se desarrolló una tercera aplicación web MVC que funciona como puente entre las otras dos (Imagen 20). En las siguientes secciones se introducen los objetivos, funciones y principales aspectos de diseño de estas aplicaciones.

Entorno Interactivo

El Entorno Interactivo es la aplicación que permite crear modelos GPSS válidos, tanto desde el punto de vista sintáctico como del modo en que se utilizan las entidades de GPSS. Esta aplicación fue desarrollada sobre tecnologías web, esto es, se ejecuta enteramente dentro de un navegador web. Su estructura modular fue diseñada para simplificar la extensión de la aplicación, tanto para incluir nuevas entidades y sentencias de GPSS como para mejorar o aportar nuevos mecanismos para brindar asistencia, soporte o ayudas a los usuarios.

Desde el punto de vista tecnológico, esta aplicación fue construida casi exclusivamente con herramientas desarrolladas sobre el lenguaje de programación Javascript. En un primer punto, y a fin de asegurar y soportar un diseño modular y una fácil extensión, se incluyó la librería RequireJS[9] en el núcleo principal de la aplicación. RequireJS es un cargador de archivos y módulos para Javascript, optimizado para ser utilizado dentro del navegador web, aunque su diseño lo hace apto también para otros entornos como por ejemplo Node.js [Watson12]. De manera transparente, RequireJS permite al desarrollador organizar su código asociando módulos unos con otros; gracias a esto, si se aplica un diseño modular y basado en objetos, permite mantener decenas o cientos de archivos Javascript simples y distribuir el código a fin de minimizar el acoplamiento funcional entre los módulos de la aplicación.

Otra de las herramientas centrales utilizadas en esta aplicación es jQuery[10]. Esta herramienta simplifica enormemente el desarrollo de aplicaciones y *scripts* en Javascript, y a su vez aporta una gran cantidad de nueva funciones al lenguaje Javascript tradicional. Tanto RequireJS como jQuery permiten al desarrollador abstraerse de los diferentes navegadores web existentes. En el contexto actual, en el cual se liberan constantemente nuevas versiones y surgen nuevas tecnologías web, la abstracción del

[9] RequireJS, a javascript module loader, página web: http://requirejs.org/.
[10] jQuery, write less, do more, página web: http://jquery.com/.

navegador resulta primordial para cualquier desarrollador web.

Entre las principales características del Entorno Interactivo se encuentra el uso de diálogos para la carga de sentencias GPSS, vistas diferentes para dichas sentencias, y paneles para la visualización de los cambios en el modelo. Para ello, los diferentes elementos visuales fueron desarrollados sobre la librería {{ mustache }}[11], en su versión para Javascript. {{ mustache }} es una librería que permite desarrollar plantillas simples y carentes de lógica dentro; esto significa que las plantillas desarrolladas con ella sólo se limitarán a mostrar la información que reciben en forma de arreglos u objetos, pero no permite realizar cambios sobre los datos originales ni tampoco ofrece condicionales o iteraciones, solo posee etiquetas. Algunas etiquetas se reemplazan con un valor, algunas no, y algunas con una serie de valores. Para generar estos valores, es posible invocar métodos y atributos de objetos de manera transparente para el programador. {{mustache }} tiene la capacidad de aplicar transformaciones muy simples sobre objetos para generar una salida, y puede utilizarse para generar código HTML así como también una gran cantidad de lenguajes de programación como Ruby, JavaScript, Python, Erlang, PHP, Perl, Objective-C, Java, .NET, Android, C++, Go, node.js, entre otros. Finalmente, entre las principales ventajas que posee {{ mustache }} se encuentra la posibilidad de organizar las plantillas en *partials*, con lo cual es posible incluir una plantilla dentro de otra. Esta característica permite componer elementos visuales muy complejos y avanzados de manera simple; además, facilita el reuso de componentes y mantiene las plantillas simples y fáciles de gestionar.

[11] {{ mustache }} Logic-less templates, página web: http://mustache.github.com/.

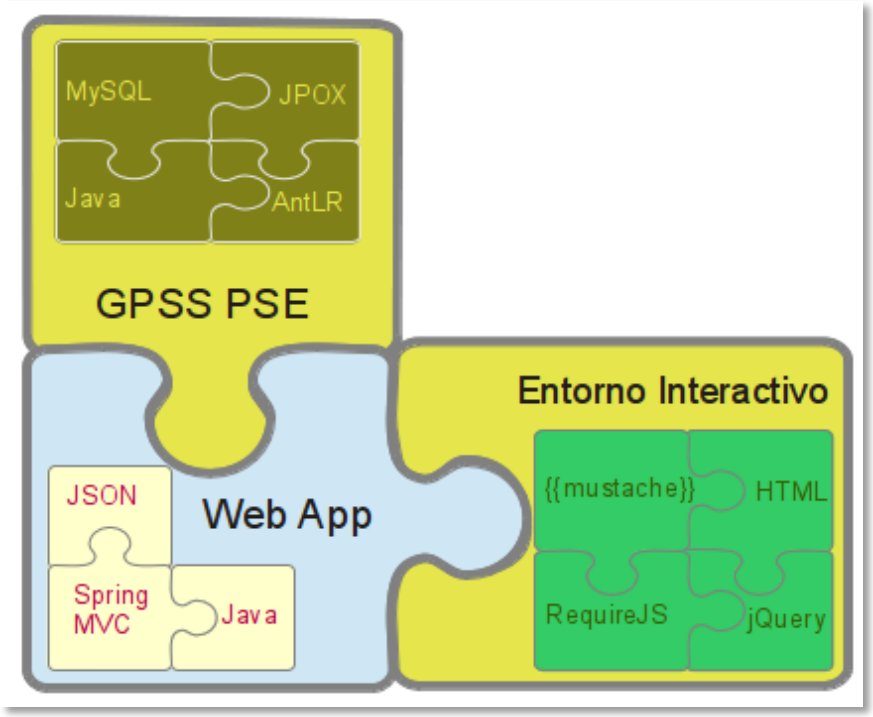


Imagen 20: Módulos y librerías que componen GPSS Interactivo

La combinación de estas librerías —jQuery, RequireJS y {{ mustache }}— permite mantener una metodología de desarrollo de RIA (Rich Internet Applications) similar a la que se aplica en otros entornos tradicionales como es el caso de aplicaciones Java o .Net. [Rosales11]. Para finalizar el listado de herramientas, además de RequireJS, jQuery y {{mustache }}, esta aplicación incluye otras aplicaciones como jQueryUI[12], jQuery Cycle y jQuery Tootlip. Es importante destacar que todas las herramientas utilizadas poseen licencias libres y abiertas: MIT, GPL versión 1 y GPL versión 2.

Diseño y módulos

El desarrollo del Entorno Interactivo para GPSS mantiene un diseño modular basado en jerarquías de objetos, y sigue el patrón de diseño *Model-View-Controller* en el cual el modelo está compuesto por la jerarquía de clases que conforman las entidades GPSS, la vista está compuesta por un conjunto de plantillas y algunas clases que permiten cargar los diferentes componentes visuales, y el *controller* por un conjunto de clases encargadas de asociar eventos de la vista con el modelo, procesar formularios y generar los objetos

[12] jQuery User Interface, página web: http://jqueryui.com/.

que las plantillas mustache requieren para mostrarse en el navegador (Imagen 21).

Por el lado del **modelo** de la aplicación, el diseño cuenta dos grandes módulos: un *metamodelo* y un *modelo dinámico*. El modelo dinámico se compone de una jerarquía de clases que replica una versión simplificada de las entidades de GPSS. Estas clases se corresponden con las instancias concretas de cada elemento creado por el usuario —de allí el término dinámico— a partir de los diálogos y componentes visuales que se le presentan. Cada entidad en esta jerarquía conforma a su vez a una meta-entidad, esto es, una entidad abstracta que pertenece al metamodelo (módulo *Metamodel*) y que describe todas las características de las entidades concretas que los usuarios pueden manipular (Imagen 22).

Las meta-entidades son utilizadas por las clases que generan diálogos para configurar correctamente los mismos: muestran parámetros, textos y ayudas en imágenes. Para ello, todas estas meta-entidades pertenecen a una jerarquía de clases cuya superclase (*Metaentity*) define todos los atributos que cada meta-entidad deberá tener como mínimo (nombre, conjunto de parámetros, etcétera). A su vez, las meta-entidades que representan los distintos tipos de bloques que el usuario puede crear están organizadas en una subjerarquía de clases, a partir de la clase MMBlock, que agrega a todos los meta-bloques la pertenencia a un determinado grupo (bloques para control de flujo, para facilidades, para almacenes, etcétera). El mismo mecanismo de subclasificación se ha utilizado para aquellos bloques que requieren operadores lógicos (menor o L, menor o igual o LE, igual o E, mayor o G, y mayor o igual o GE) u operadores condicionales (facilidad no disponible o FNV, almacén no vacío o SNE, entre otros).

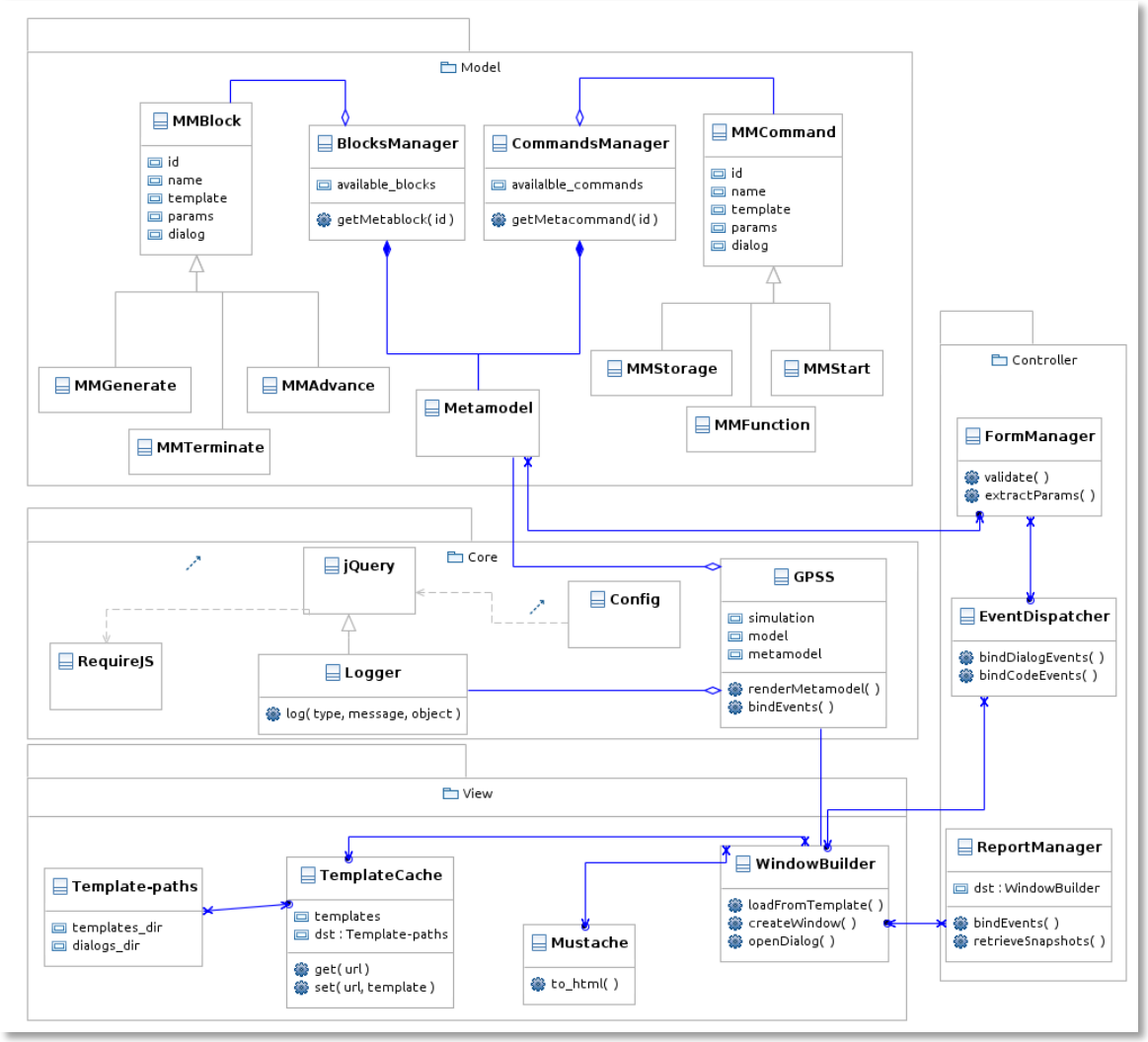


Imagen 21: Principales módulos y clases de GPSSi

Para trabajar las distintas meta-entidades disponibles (localizarlas, agruparlas, acceder a sus atributos) se incluyeron dos clases gestoras: *CommandsManager* para los meta-comandos, y *BlocksManager* para los meta-bloques. Estas clases funcionan como *helpers,* puesto que su rol es principalmente proveer funcionalidad sobre las meta-entidades y evitar así ensuciar su código con operaciones relacionadas con su gestión.

Existe también una clase llamada *FormManager*, que pertenece al módulo Controller, y que utiliza la información contenida en los atributos de las meta-entidades para procesar los datos ingresados por el usuario, validarlos y enviarlos hacia el modelo, a través de los diferentes EntityManagers, para que generen las diferentes entidades del modelo.

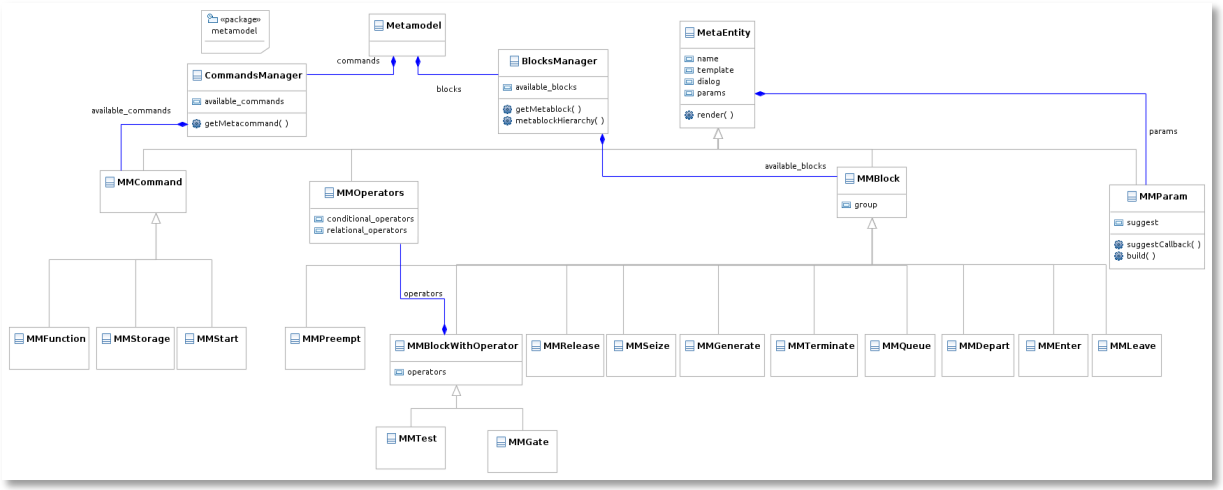


Imagen 22: Clases del metamodelo de GPSSi

Como se mencionó previamente, en el modelo dinámico se generan las entidades que los usuarios manipulan. Para ello, el módulo *Model* cuenta con varias clases *entity*, que representan a las entidades que los usuarios crean y eliminan de sus modelos (colas, comandos, bloques, etc.), y otras clases *entitiy-managers*, responsables de generar instancias de dichas entidades, de incorporarlas al modelo GPSS que está siendo construido, y de gestionarlas durante el ciclo de vida del modelo (cambio de ordenamiento, eliminación). Existe pues una jerarquía de clases cuya clase superior es EntityManager y que posee una subclase CommandsManager a cargo de la gestión de todos los comandos, y otra subclase BlocksManager encargada de gestionar todos los bloques. El modelo se completa con otras subclases para la gestión de otras entidades permanentes que no se corresponden directamente con sentencias de GPSS (bloques y comandos): FacilityManager para facilidades (*facilities*), StorageManager para almacenes (*storages*), QueueManager para colas (*queues*), y así sucesivamente. La clase abstracta EntityManager implementa los métodos para construir entidades y asociarlas al modelo, y todas sus subclases heredan todo su comportamiento, y su único rol es instanciar la clase correspondiente con los parámetros correctos.

El mecanismo arriba descripto de entidades y meta-entidades puede parecer algo complicado a simple vista, pero posee algunas características muy importantes:

- la generación del espacio de trabajo para el usuario se realiza de manera automática, por medio de la exploración de las meta-entidades que representan bloques y comandos, la detección de los diferentes grupos, y la discriminación

según el tipo (bloque o comando);

- la incorporación de nuevos bloques o comandos para trabajar con entidades ya existentes resulta muy simple, pues sólo se precisa configurar la meta-entidad (definir parámetros y asignar grupo), y del resto se encargarán los distintos *entity managers*;
- la creación de nuevas entidades de cero junto a sus managers asociados y los bloques y comandos requeridos para trabajar con la misma se mantiene también como tarea sencilla. Esto se debe a que las clases de las distintas entidades, así como también los metabloques y metacomandos son principalmente descriptivos y poseen muy poca lógica, y los managers de meta-entidades sólo requieren indicar sobre qué clases trabajan;
- la actualización de la vista en base a los cambios se realiza de manera automática gracias a la implementación de una versión en Javascript del patrón de diseño *Observer (*esto se detalla más adelante) y la habilidad de los managers de entidades de detectar cambios en el modelo y notificarlos.

Por el lado de la **vista**, el Entorno Interactivo para GPSS está compuesto por un conjunto de clases que permiten, por un lado representar los distintos espacios del área de trabajo, y por el otro que se encargan de realizar las operaciones necesarias para generar los diferentes elementos visuales que se le presentan al usuario (diálogos, ayudas textuales, imágenes, etc.). La clase central del módulo *View* se llama WindowBuilder, y su rol principal es el de construir los elementos visuales y asociar eventos dinámicos (al abrirse, mostrarse y cerrarse, al interactuar con el ratón y el teclado, etcétera). Esta clase está ligada con la librería Mustache, que como se explicó más arriba, permite generar las vistas HTML de los diferentes elementos a partir de ciertas plantillas predefinidas. A su vez, la clase *AppConfig* —que mantiene toda la configuración de la aplicación web— colabora con la clase WindowBuilder para indicarle los directorios y archivos en el servidor dónde se encuentran las distintas plantillas (*templates*).

Las distintas vistas HTML de todos los elementos visuales se componen de plantillas generales que a su vez se componen de plantillas más específicas; de manera similar, estas plantillas se componen de otras plantillas aún más específicas, y así sucesivamente.

Por ejemplo, para visualizar los bloques disponibles para trabajar con facilidades, existe una plantilla que muestra un conjunto de grupos de bloques, junto con la cantidad de bloques de cada grupo, y con el efecto visual *roller*; cada uno de estos grupos se genera mediante otra plantilla que muestra todos los bloques disponibles bajo cierto grupo; a su vez, cada uno de estos bloques se compone de otra plantilla que muestra el nombre de cada bloque y la lista de parámetros para ese bloque; finalmente, cada uno de estos parámetros se genera a partir de una plantilla *ad hoc* (Imagen 23).

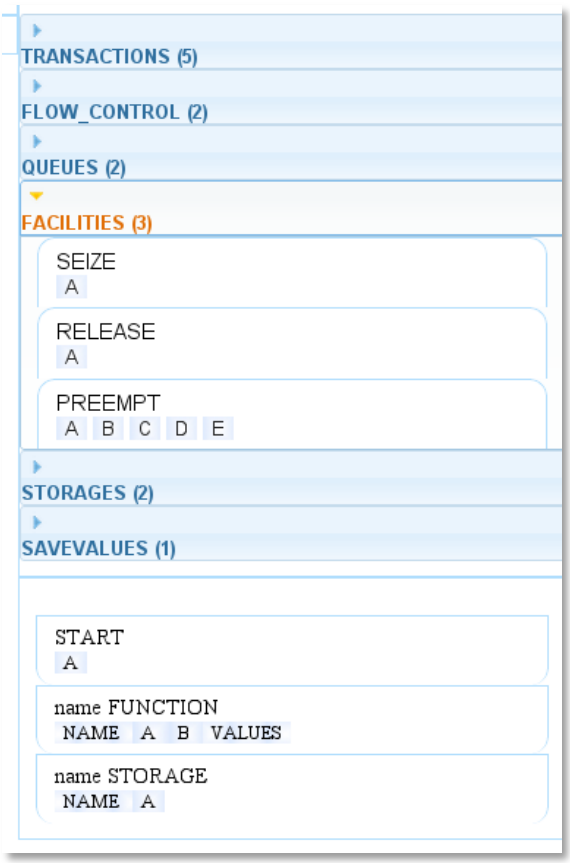


Imagen 23: Agrupación por categorías de bloques y lista de comandos disponibles

La composición de plantillas simplifica la gestión de los documentos .html dado que estos se mantienen simples y enfocados en *mostrar* elementos muy específicos, y simplifica enormemente el reuso, mejora y/o reemplazo de componentes visuales. Por ejemplo, existe una única plantilla encargada de generar la vista de un parámetro; esta plantilla es utilizada para generar todos los parámetros de todos los bloques y comandos disponibles para el usuario. De esta forma, si se requiere generar una nueva forma de mostrar estos parámetros, o adicionar alguna funcionalidad especial (por ejemplo,

previsualización de los mismos al pasar el ratón por encima), solo será necesario alterar esta única plantilla.

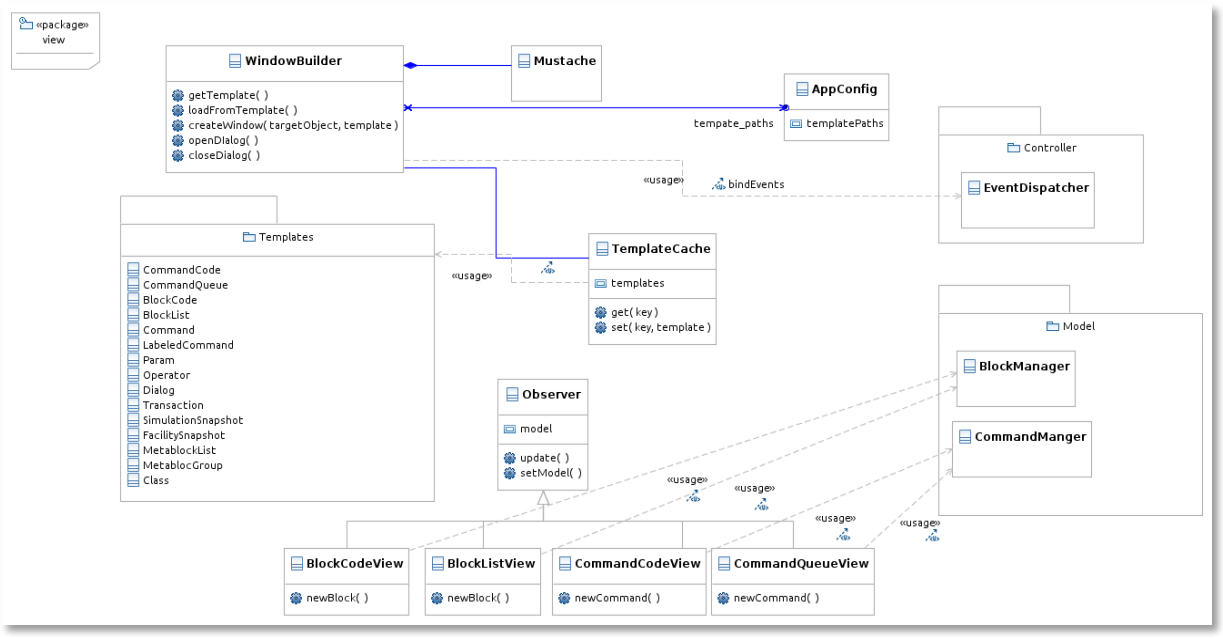


Imagen 24: principales clases del paquete View

Las plantillas son recuperadas desde el servidor a medida que el usuario las requiere. Esto ahorra mucho tiempo de carga inicial de la aplicación web, puesto que sólo se envían a través de la red aquellas plantillas que realmente se mostrarán. Por ejemplo, no tiene sentido enviar al cliente durante la carga de la aplicación las plantillas que muestran resultados de una simulación. Sin embargo, una vez que una plantilla ya ha sido enviada hacia el navegador web del cliente, no tiene sentido que vuelva a solicitarse al servidor en caso de que la misma plantilla sea requerida. Muchos navegadores incluyen un mecanismo de caché, que debería ayudar a resolver este problema; sin embargo, por tratarse de *requests* AJAX dinámicos, los mecanismos de caché de los navegadores no siempre funcionan como se espera. Para evitar la transmisión innecesaria de plantillas, se implementó un mecanismo de memoria temporal (caché) interno —representado por la clase *TemplateCache*— cuyo rol es mantener un registro de las plantillas ya recibidas en el cliente, y realizar solicitudes AJAX sólo cuando se trata de plantillas nunca enviadas (Imagen 24). Para implementar este mecanismo de caché, a cada plantilla se le asocia una clave y se almacena en una colección interna a dicha clase,

lo que permite recuperarlas muy fácilmente.

Como se mencionó previamente, el espacio de trabajo se dividió en cuatro paneles que muestran el estado del modelo y se actualizan a medida que el modelo cambia (Imagen 25). Estos paneles son:

- ✓ ***Panel de comandos en modo código GPSS*** (clase *CommandCodeView*): toma las entidades de tipo *Command* que el usuario ha creado y los muestra en forma de código GPSS. Se encarga, por ejemplo, de separar en dos líneas los comandos que así lo requieren (ej. *Function*).
- ✓ ***Panel de comandos en modo cola*** (clase *CommandQueueView*): muestra las entidades de tipo *Command* creadas por el usuario, pero en una vista de cola de comandos. Como se verá más adelante, esta vista tiene como objetivo destacar el ordenamiento FIFO de los comandos. Este panel permite, por ejemplo, organizar la cola de comandos mediante el mecanismo de arrastrar y soltar (*drag & drop*).
- ✓ ***Panel de bloques en modo código GPSS*** (clase *BlockCodeView):* al igual que el panel de código de comandos, este panel muestra las entidades de tipo *Block* creadas por el usuario. En este tipo de vista se resaltan con un color las palabras claves del lenguaje (*seize*, *preempt*, *generate*, etc.), y se destacan con otro color las entidades con las que cada bloque trabaja (las facilidades, colas, almacenes, etc.). Esta vista agrega también unos pequeños íconos a cada bloque, que permiten al usuario subir, bajar o eliminar bloques directamente desde el espacio de código.
- ✓ ***Panel de bloques en modo lista*** (clase *BlockListView*): este panel muestra los bloques como una lista secuencial, y agrega funciones de ordenamiento de los bloques similares al panel de vista de la cola de comandos.

Dentro del módulo que contiene al modelo, existen dos clases que se encargan de gestionar los elementos de GPSS que el usuario crea, modifica y elimina, o sea el modelo GPSS en concreto. Estas clases son *BlocksManager* y *CommandsManager*. Cada vez que el modelo es alterado, se realiza a través de estas clases; a su vez, los distintos paneles están asociados a estas clases bajo un mecanismo similar al patrón de diseño *Observer* [Gamma94].

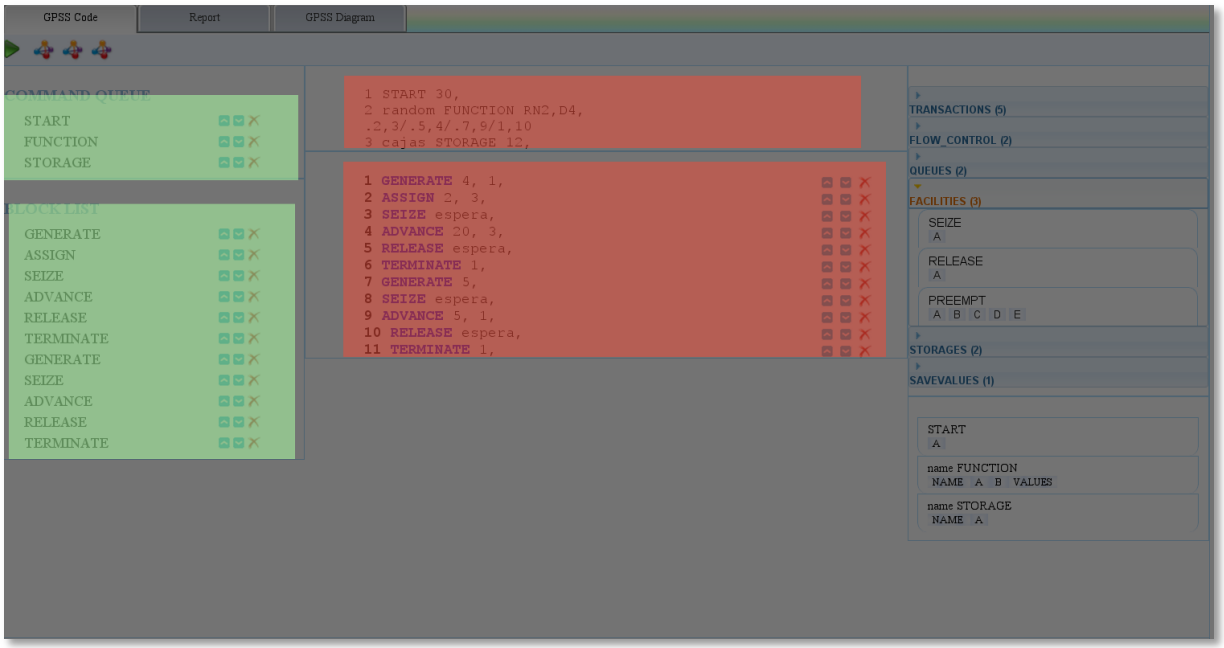


Imagen 25: Paneles observadores del Entorno Interactivo:
zonas de código (rojo) y zonas de vista de listas y colas (verde)

Los paneles que muestran las distintas vistas de los comandos poseen dos clases encargadas de gestionarlos y actualizarlos, llamadas *CommandCodeView* y *CommandQueueView*. Estas clases a su vez se asocian con la clase *CommandsManager* que conoce todos los comandos que posee el modelo hasta el momento. De manera similar, los paneles que muestran las vistas de los bloques poseen sus clases encargadas de gestionarlos y actualizarlos, llamadas *BlockCodeView* y *BlockListView*, quienes a su vez están asociadas con la clase *BlocksManager*. De este modo, al crearse, por ejemplo, un nuevo bloque en el modelo, la clase *BlocksManager* crea la entidad Block, la cual debe luego registrar en el modelo en la posición correspondiente, y a continuación informar a sus *observadores* que ha habido un cambio en los bloques del modelo. Estos observadores analizan el nuevo estado del modelo, y muestran en pantalla —cada uno según el formato para el cual ha sido creado— el modelo actualizado.

Este mecanismo de *Observer* puede parecer algo intrincado al principio, pero posee algunas ventajas entre las que se destacan:

1. resulta muy flexible para mostrar la misma información de manera diferente, o lo que es lo mismo, mantener distintas vistas de la misma entidad según el propósito buscado (mostrar código fuente, destacar ordenamiento);

2. es fácilmente ampliable y puede generar vistas para otras clases del modelo, como por ejemplo *FacilityManager* o *StorageManager*, a fin de mostrar al usuario alguna representación de las facilidades o almacenes que han sido creadas;

3. permite generar otras vistas alternativas de los distintos elementos del modelo, como una vista basada en formas geométricas y conexiones entre las mismas que represente cómo se conectan los bloques y entidades del modelo.

Finalmente, por el lado del **controlador**, el Entorno Interactivo para GPSS cuenta con la clase *EventDispatcher*, que se encarga de asociar diversos eventos entre los elementos de la vista y ciertas acciones sobre elementos del modelo durante la etapa de creación de modelos GPSS. Estos eventos abarcan tanto a diálogos y ventanas como a las distintas vistas de las entidades creadas dinámicamente (bloques y comandos). Durante la etapa de análisis de simulaciones previamente ejecutadas, existe una clase ReportManager que asocia eventos entre los elementos visuales del reporte o la vista de un instante de simulación con los elementos del modelo correspondientes con dicha simulación. La clase *ReportManager* también recibe los eventos del ratón sobre la vista de un instante de simulación y recupera desde el servidor toda la información relativa al elemento que recibió el evento. Por ejemplo, cuando el usuario hace un clic sobre un tiempo de reloj de simulación, se ejecuta la función *retrieve_snapshot* que se conecta con el servidor remoto, le solicita toda la información relativa a esa simulación en ese instante de reloj de simulación, y procesa el paquete de resultados enviados por el servidor para luego separarlo en partes más pequeñas (listado de facilidades activas, cadena de eventos actuales, cadena de eventos futuros, etcétera) y finalmente enviar al módulo Vista cada una de estas partes a fin de que las mismas sean mostradas al usuario mediante el mecanismo de plantillas descripto previamente.

En resumen, el Entorno Interactivo de GPSS es una herramienta web que puede ser ejecutada en modo *standalone*, esto es, sin la necesidad de un servidor web ni un servidor de aplicaciones, y que fue diseñada con fines didácticos bajo un esquema de módulos y clases simples y de fácil manutención. Todos los módulos fueron pensados para ser ampliados, ya sea para incorporar nuevas entidades al modelo GPSS (en el módulo *Model*), como para incluir nuevas herramientas didácticas que realicen algún aporte específico durante la enseñanza del simulador GPSS, como vistas de nuevas

entidades, otros mecanismos de asistencia y ayuda en línea, nuevas funciones de exploración de resultados de simulación, etcétera (módulos *Controller* y *View*).

Motor de Simulación de GPSS con persistencia

La segunda aplicación que compone GPSS Interactivo es el Motor de Simulación de GPSS con persistencia (GPSS Persistent Simulation Engine, o simplemente GPSS PSE). Esta aplicación tiene como objetivo principal ayudar a los alumnos/usuarios a interpretar cómo funciona el motor de simulación de GPSS por dentro, lo que significa comprender cómo se ejecutan los modelos, cómo se alteran las entidades permanentes y cómo interactúan unas con otras. Para ello, GPSS PSE realiza básicamente tres operaciones:

1. ejecuta modelos desarrollados en GPSS;
2. mantiene un historial de cambios sobre los modelos que ejecuta;
3. recupera total o parcialmente de los elementos del historial.

En las siguientes secciones se detallan los principales aspectos de diseño e implementación de GPSS PSE que permiten realizar estas 3 funciones.

Modelo del simulador GPSS

GPSS PSE replica en su modelo interno una versión simplificada del modelo original del simulador de GPSS. La expresión "versión simplificada" hace alusión al hecho de que no se ha replicado la totalidad de las entidades y sentencias definidas en GPSS, sino que se ha seleccionado un subconjunto de ellas que permiten el uso del motor de simulación con fines educativos. Las entidades seleccionadas sirven, como se verá más adelante, para que los alumnos/usuarios comprendan cómo es el flujo de transacciones dentro del motor de simulación, y cómo influyen los distintos tipos de entidades permanentes en los resultados obtenidos.

El modelo de GPSS PSE está integrado por una jerarquía de clases que representa a las distintas entidades de GPSS, y que está encabezada por la clase abstracta Entity. Esta clase posee el mínimo conjunto operaciones y atributos que cualquier entidad GPSS

posee: los atributos ID (identificador) y RetryChain (cadena de reintentos, o sea, cadena que aloja a todas las transacciones que *observan* el estado de la entidad), y operaciones para alterar su estado, registrar observadores y evaluar expresiones y atributos estándares del sistema o SNA. Dentro de de esta jerarquía (Imagen 26) se incluyen las clases:

- *BlockEntity*, que abarca a todos los bloques de GPSS, implementados en subclases como *GenerateBlock*, *SeizeBlock*, *AdvanceBlock*, etc.;
- *CommandEntity*, que representa a los comandos de GPSS;
- *SystemClock*, que representa al reloj de simulación de GPSS;
- *TransactionEntity*, que representa a las entidades temporales —transacciones— de GPSS;
- *StorageEntity*, que representa a los almacenes;
- *FacilityEntity*, que representa a las facilidades;
- *RandomNumberGeneratorEntity*, que representa a los 8 generadores de números aleatorios de GPSS;
- *SimulationEntity*, que representa a la simulación;
- *ChainEntity*, que abarca a las cadenas globales de GPSS (Currrent Events Chain y Future Events Chain), a las cadenas específicas de ciertas entidades (Delay Chains, Retry Chains, Interrupt Chains y Pending Chains) y cadenas definidas por el usuario (User chains).

Todas estas clases son instanciadas de acuerdo al modelo que ha sido creado por el usuario, a partir del código GPSS. Dada las características propias de GPSS, algunas entidades son creadas al momento de ejecutar la cola de comandos (funciones, almacenes, reloj) mientras que otras son creadas dinámicamente al momento de ser invocadas por primera vez por determinados bloques (facilidades, colas, cadenas de usuario). Esto quiere decir que esta jerarquía de clases representa de hecho al modelo dinámico que el usuario ha programado, el cual es alterado constantemente a medida que la simulación avanza.

Dentro de esta jerarquía, la clase *SimulationEntity* juega un rol fundamental, ya que es

la encargada de mantener el estado de la simulación en todo momento de la ejecución del modelo. Para ello, la clase *SimulationEntity* posee una colección con todas las facilidades del modelo, otra colección con todos los almacenes, otra con las colas (*QUEUES*), las cadenas FEC y CEC, el reloj del sistema, la cola de comandos y la lista de bloques. Asimismo, existe una clase adicional, llamada *TransactionScheduler*, que encapsula el comportamiento del algoritmo de planificación de transacciones de GPSS, descripto en el capitulo 3. Además de estas referencias directas, cada clase que representa a alguna entidad de GPSS posee también sus propias referencias, formando así un grafo dirigido cíclico de composición de entidades del modelo (Imagen 27). Como se verá más adelante, las conexiones entre las entidades del grafo de composición juegan un papel fundamental al momento de registrar los cambios en el modelo y mantener una historia sobre la evolución de la simulación.

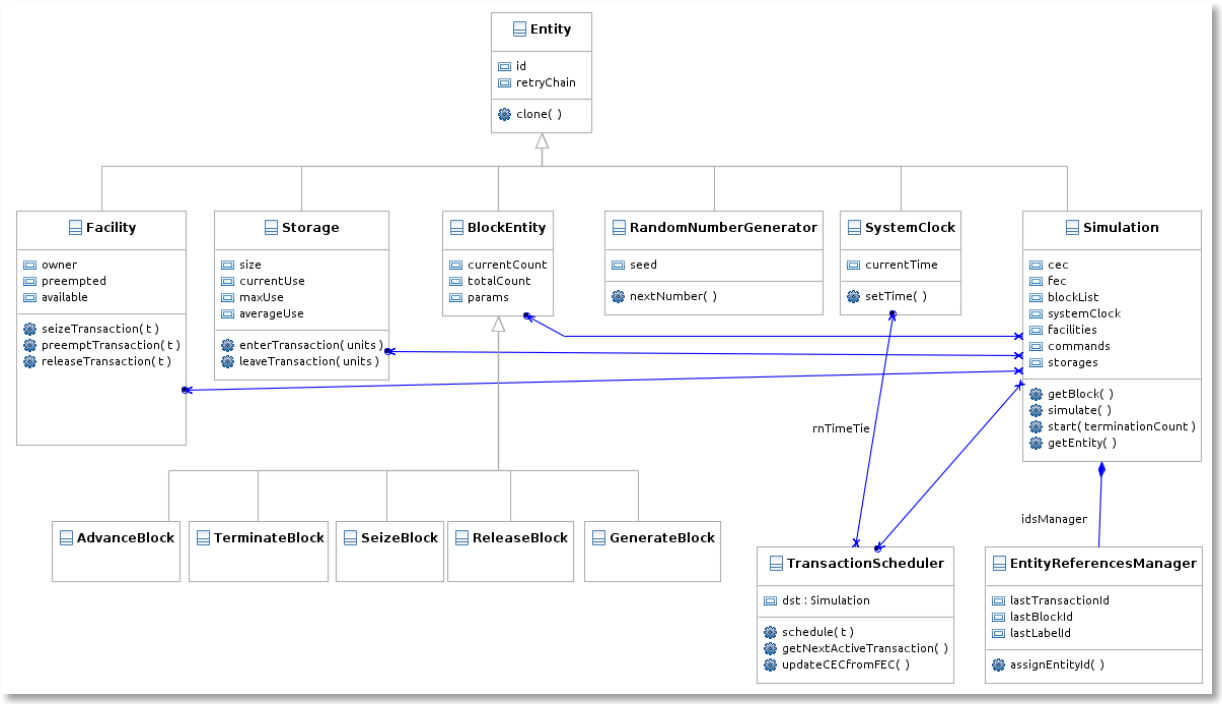


Imagen 26: Principales elementos de la jerarquía de clases de entidades del modelo

Creación y localización de entidades del modelo

El dinamismo que caracteriza a los modelos requiere la creación constante de nuevas entidades, que por lo general son transacciones pero en muchas ocasiones también serán entidades permanentes en tiempo de ejecución. Esto sucede, en particular, cuando

se referencia por primera vez a una entidad que no había sido definida y/o inicializada previamente mediante un comando. En particular, esto es muy común cuando no se especifica el nombre de una entidad, sino que se utiliza un valor numérico alojado en un atributo de una transacción. Por ejemplo:

```
...
1 GENERATE 10,2
2 ASSIGN cajero,(DUNIFORM(1,1,4))
3 QUEUE *cajero
4 SEIZE *cajero
...
```

En este ejemplo, se crean transacciones cada 10±2 unidades de tiempo; la segunda línea almacena en el atributo llamado "cajero" de la transacción un valor entre 1 y 4 con igual probabilidad. Dicho de otro modo, se crea un nuevo atributo en cada transacción que ejecuta el bloque ASSIGN (línea 2), y se le asigna el valor 1, el valor 2, el valor 3 o el valor 4; la probabilidad de asignar cualquiera de estos valores será, en todos los casos, del 25%. Luego, en la línea 3, la transacción ingresa a la cola cuyo nombre/identificador se encuentra alojado en el atributo *cajero* de la transacción, lo que significa que, por ejemplo, si dicho atributo contiene el valor 1, se estará ingresando a la cola 1. Algo similar sucede en la línea 4, en la cual la transacción intenta tomar la facilidad cuyo nombre/identificador es también el valor contenido en el mismo atributo *cajero*. A medida que el modelo se ejecuta y que la función discreta-uniforme de la línea 2 retorna valores por primera vez (valores que aún no había retornado), el modelo actual de la simulación deberá expandirse para incorporar primero una nueva cola (línea 3) y a continuación una nueva facilidad (línea 4). Al finalizar la ejecución de este modelo, podrían existir 4 colas y 4 facilidades, pero también podría haberse creado sólo 3 de ellas, o quizás 2 o incluso una sola. Esto estará determinado por los valores que genere la función DUNIFORM y por la cantidad de transacciones que logren crearse en la línea 1 (podría haber un valor de *termination count* muy bajo, o podrían existir otros generadores que se ejecuten a intervalos más cortos que el del ejemplo).

Al crear nuevas entidades, el motor de GPSS debe asignarle un identificador único de

entidad, y asociar dicho identificador con el nombre de la entidad. Es importante destacar que un mismo nombre puede estar asociado a más de una entidad, siempre y cuando las entidades sean de distinto tipo. Por ejemplo, en GPSS puede existir una cola con el mismo nombre de una facilidad, lo cual permite asociar fácilmente a la cola como entidad encargada de mantener estadísticas de espera y permanencia en la facilidad homónima. Esto mismo puede aplicarse a almacenes, cadenas de usuario, y cualquier otra entidad permanente. Sumado a esto, el programador puede utilizar el comando EQU para fijar una equivalencia con un nombre y un valor, que puede luego tomar el rol de identificar de entidad.

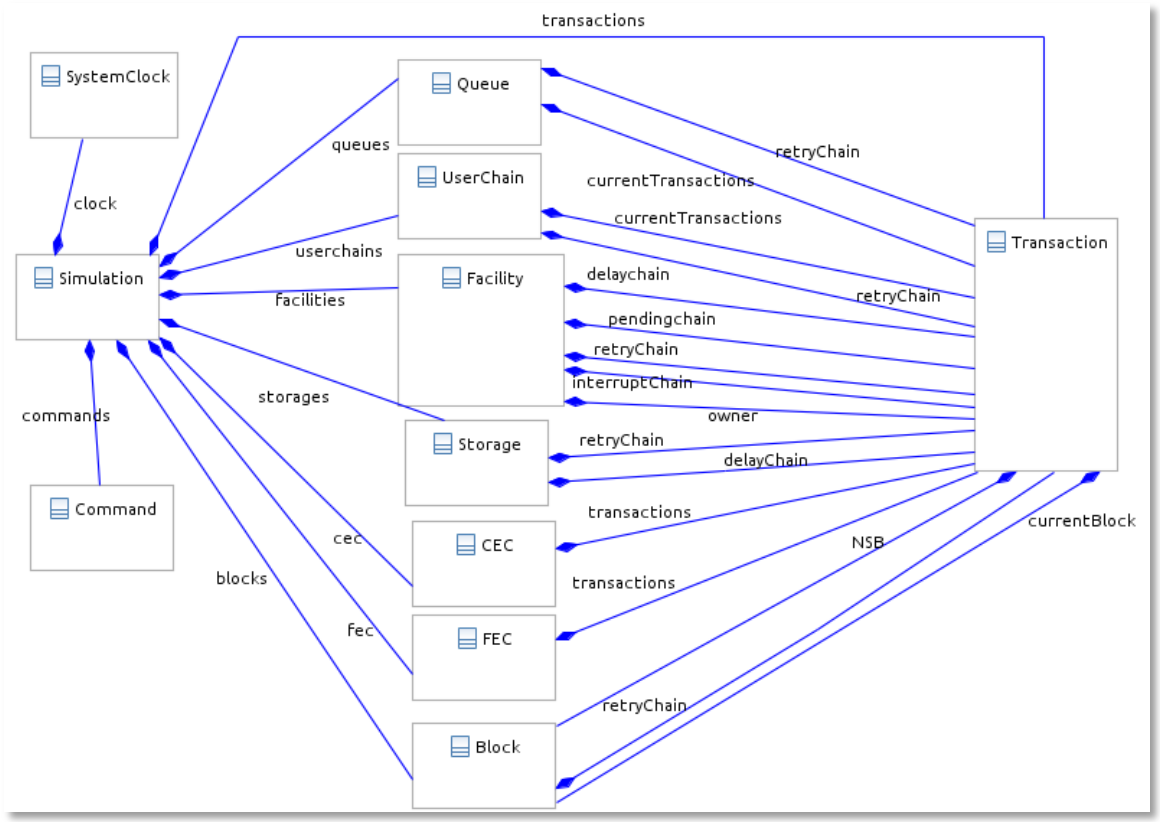


Imagen 27: Grafo de composición de las principales entidades del modelo

Este sistema de manejo de identificadores y nombres implica que el motor de ejecución debe ser capaz de crear entidades y asignarles un identificador único, y en algunos casos debe ser capaz de asociar un nombre con uno o más identificadores, algunos de ellos asignados por el simulador y otros "reservados" por el programador. Asimismo, el simulador debe realizar la tarea inversa durante la ejecución de la

simulación: al detectar una referencia a un nombre de una entidad, deberá determinar el tipo y el identificador único de la misma, para luego identificar unívocamente a la entidad en cuestión para realizar la operación o ejecutar la rutina correspondiente. Esta última tarea es de particular importancia puesto que permite al simulador localizar entidades y comunicarse con ellas durante el tiempo de ejecución de la simulación.

Para realizar estas operaciones, se ha creado una clase llamada *EntityReferencesManager*, que está directamente relacionada con la clase Simulation, y que brinda servicios de creación y alocación de entidades por un lado, y de localización y referenciamiento por el otro. Puede compararse su funcionamiento al de un servidor de nombres de dominio (DNS): cada vez que el simulador requiere trabajar con una entidad en particular y conoce su identificador, éste le solicita a la clase *EntityReferencesManager* que encuentre a la entidad a partir del identificador de entre todas las entidades que viven en el modelo en ese momento. Esta clase localiza a la entidad en cuestión y retorna una referencia a la misma, lo que le permite al simulador enviarle mensajes para consultar su estado actual o para ejecutar sus rutinas internas.

Tipos de cadenas

Como ya se mencionó en secciones anteriores, durante la ejecución de simulaciones en GPSS, las transacciones sólo pueden encontrarse en alguna cadena del sistema, con la única excepción de la transacción activa, una sola en todo momento. Existen cadenas globales, como la Cadena de Eventos Actuales (CEC, por *Current Events Chain*) y la Cadena de Eventos Futuros (FEC, por *Future Events Chain*), y cadenas de las distintas entidades. Todas las entidades permanentes poseen como mínimo una cadena, la cadena de reintentos o *RetryChain*. Algunas entidades poseen una cadena de transacciones en espera o *DelayChain* (facilidades, almacenes); las facilidades poseen además dos cadenas adicionales, para transacciones que intentan un preempt (*PendingChain*) y transacciones que perdieron la facilidad por medio de un preempt (*InterruptChain*). Además, existe una entidad particular que permite al usuario definir sus propias cadenas, o sea un espacio adicional para almacenar transacciones junto a un mecanismo personalizado para quitar las transacciones de sus cadenas y reingresarlas al modelo.

El propio diseño del lenguaje hace de las cadenas ciudadanos de primera clase dentro

de los modelos, ya que son utilizadas en todas las entidades permanentes al menos una vez. Las cadenas, más allá de su tipo, comparten en todos los casos un funcionamiento común que consiste en alojar transacciones y removerlas de acuerdo a diferentes criterios; estos criterios determinan el propósito de cada cadena. Por ejemplo, la cadena de eventos futuros almacena las transacciones ordenadas a partir del tiempo de partida (BDT, por *Block Departure Time*) y, al momento de removerlas, selecciona aquellas transacciones cuyo BDT sea menor o igual al tiempo actual del reloj de simulación; la cadena de reintentos, por otro lado, organiza a las transacciones según el orden de llegada (FIFO) únicamente, y al momento de removerlas selecciona la totalidad de transacciones que contiene y las envía a la CEC; como último ejemplo, la CEC organiza a las transacciones según su prioridad (las de mayor prioridad primero) y en orden FIFO para todas las transacciones de igual prioridad; al momento de removerlas, selecciona transacciones de a una, siempre la primera de la cola, y la coloca como transacción activa de la simulación.

Dentro del diseño de GPSS PSE se añadieron dos jerarquías de clases para representar las distintas cadenas del sistema. Por un lado, se modelaron las cadenas cuyo orden está determinado por el orden de llegada de las transacciones, hasta ahora FIFO y LIFO, bajo la superclase de Java *LinkedList*. Por el otro lado, se modelaron las cadenas cuyo orden depende de algún atributo de las transacciones, como ser el BDT o la prioridad (PR), bajo la superclase *SortedList*, implementada dentro del paquete *Utils*. En ambos casos, las cadenas deben implementar la interfaz *TransactionChain*, lo cual asegura que, sin importar el tipo de cadena de la cual se trate, será capaz de ejecutar las operaciones mínimas esperables:

- *getFirst*(): retorna el primer elemento de la cadena, según el orden correspondiente;
- *getFirsts*(): retorna una lista de todas las transacciones consideradas primeras (especialmente útil para cadenas que mueven grandes conjuntos de transacciones, como *RetryChain*, FEC o *UsersChains*);
- *removeFirst*(): remueve y retorna la primer transacción de la cadena, según el orden que corresponde.

Además, esta interfaz posee la capacidad de comparar transacciones según algún

criterio (prioridad, BDT), lo cual es utilizado al momento de incorporar nuevas transacciones a las cadenas, asegurando así respetar el orden correspondiente.

El caso de la *RetryChain* presenta una situación especial, ya que todas las entidades permanentes poseen una cadena de este tipo. Estas cadenas se utilizan para alojar todas las transacciones que estén esperando que el estado de las entidades que las contienen cambie de algún modo. Esto sirve, principalmente, para bloquear transacciones en bloques condicionales como *GATE* y *TEST*, en los cuales las transacciones quedan detenidas hasta que la condición que está siendo evaluada se vuelva verdadera. Por ejemplo:

```
...
1 TEST SF$cajas
2 ENTER cajas
...
3 LEAVE cajas
...
```

En este ejemplo, en la línea 1, se utiliza el atributo numérico del sistema (SNA) llamado *Storage Full* (almacén completo o SF), cuyo valor será 1 (verdadero) en caso que el *storage* cajas esté completo, y 0 (falso) en caso que aún le quede algún espacio libre. Una vez que el almacén se ha completado, todas las transacciones que arriben al bloque TEST de la línea 1 serán bloqueadas allí, a la espera de que el estado del almacén *cajas* sea modificado. Esto significa que estas transacciones ingresarán a la *RetryChain* de dicho almacén, y quedarán detenidas allí hasta que algún evento lo modifique. En algún momento, una transacción activa ejecutará el bloque 3, y liberará un espacio del almacén. Este evento provocará el movimiento de todas las transacciones que se encontraban en la *RetryChain* de *cajas* hacia la CEC, para así poder reevaluar el *TEST* y continuar la ejecución normal de sus bloques.

El ejemplo arriba descripto, aplicado a un almacén en particular, puede repetirse para cualquier entidad permanente cuyo estado pueda ser consultado mediante un SNA: facilidades, colas, cadenas de usuario, bloques e incluso la simulación misma. Es por esto

que se implementó un mecanismo genérico que asegure que este comportamiento común pueda replicarse en todas las clases que se corresponden con entidades permanentes. Este mecanismo funciona —nuevamente— de manera similar al patrón de diseño *Observer*: cada vez que una transacción ingresa a la *RetryChain* de una entidad, se registra mediante el método *addObserverTransaction()* como "observadora" de esta entidad. A su vez, cada vez que el estado de una entidad cambia, la entidad invoca al método *StateChange()*, lo que provoca que todas las transacciones observadores se activen nuevamente. Vale la pena aclarar aquí que el método StateChange se invocará cualquiera sea la variable de estado que ha cambiado, pues a priori el simulador no tiene forma de saber cuál fue el motivo por el cual las transacciones de la *RetryChain* han sido enviadas allí: mientras que algunas ingresaron esperando que el SNA SF sea verdadero, otras podrían haber ingresado desde un bloque *TEST SNV$almacen*, o sea esperando que el almacén pase a estar deshabilitado temporalmente.

Evaluación de expresiones en GPSS

Como ya se ha visto, GPSS incorpora la idea de atributo numérico del sistema, o simplemente SNA (*Standard Numeric Attribute*). Un SNA no es más que una expresión que permite evaluar y retornar algún atributo de una entidad permanente en particular. Los SNA son utilizados muchas veces para formar expresiones complejas en GPSS, las cuales también pueden incluir valores numéricos o nombres de entidades (equivalentes a otros valores), referencias a atributos de las transacciones, invocación a funciones del sistema, y por supuesto distintos tipos de operadores como suma, resta, multiplicación y división.

GPSS representa un caso muy especial al momento de evaluar expresiones aritméticas, ya que el intérprete del lenguaje debe ser capaz en primera medida de discriminar los distintos elementos posibles según el tipo de cada uno, y luego de invocar a la función o rutina en caso de que corresponda hacerlo. Adicionalmente, algunas expresiones compuestas por SNA indican la entidad desde la cual deberá evaluarse el atributo, mientras que otras poseen dicha entidad implícita. Por ejemplo, los siguientes SNA no requieren especificar la entidad en cuestión:

- XN1: retorna el número de la transacción activa.

- *X : atributo X de la transacción activa (X puede ser un nombre o un número).
- P2 o P$2: parámetro 2 de la transacción activa. En algunos casos GPSS requiere que se utilice el símbolo $, en otros el símbolo *, y en otros ningún símbolo en particular.
- AC1: valor absoluto del reloj del sistema, desde la ejecución del último comando CLEAR.
- C1: valor relativo del reloj del sistema, desde la ejecución el último comando RESET.
- TC1: valor del *TerminationCount* de la simulación.

Por otro lado, algunos SNA no tienen sentido sin una entidad sobre la cual aplicarse, por ejemplo:

- FN$obtenerValor: indica que debe evaluarse la función *obtenerValor* y retornarse su valor.
- V$calcularMonto: provoca la evaluación de la expresión contenida en la variable *calcularMonto*, y retorna su valor.
- F$cajero: evalúa si la facilidad *cajero* se encuentra ocupada, en cuyo caso retorna 1. Si esta facilidad estuviese libre, el SNA F retornará 0.
- FC$cajero: calcula el número de veces que la facilidad *cajero* ha sido tomada por alguna transacción, ya sea mediante un bloque SEIZE o mediante un bloque PREEMPT, y retorna dicho valor.
- N$reintentar: calcula el número total de transacciones que han ingresado al bloque con la etiqueta *reintentar*.

A partir de los ejemplos previos, es posible observar que la acción de evaluar un SNA puede requerir primero localizar alguna entidad en particular a partir de su nombre/etiqueta o su identificador, y en algunos casos puede requerir ejecutar alguna rutina interna para obtener un resultado. Por ejemplo, para ejecutar el SNA F (Facility Busy), el simulador deberá consultar el estado de la facilidad, o sea analizar si la misma posee una transacción *owner*, y retornar un valor (0 o 1) según corresponda. De todos modos, un SNA se compone por lo general de un código (P, F, AC1, XN1) y en algunos

casos de un identificador de entidad (*obtenerValor*, *calcularMonto*, etc.), mientras que en otros casos la entidad se obtiene implícitamente a partir del contexto (SNA de la entidad Simulation o de la transacción activa, por ejemplo).

Para soportar los distintos casos de SNA, el modelo de GPSS PSE incluye una clase especial llamada *SNAMapping*, cuyo rol principal consiste en identificar la clase de la entidad a la que hace referencia cada SNA. Por ejemplo, para el SNA FC esta clase indicará que se está trabajando con una entidad de clase *Facility*, mientras que para el SNA *P* (*transaction parameter*) se estará hablando de una *Transaction*. Una vez localizada la clase de la entidad, se aprovecha la capacidad de la clase *EntityReferencesManager* para localizar entidades, y se le solicita que localice la entidad de la clase en cuestión, y con el identificador especificado en la expresión, salvo que se trate de una entidad global como *Simulation*, que no requiere ser localizada por esta clase. Finalmente, sólo resta conocer cuál es la operación o subrutina sobre la entidad localizada que deberá ejecutarse. Para ello, todas las entidades de GPSS son "conscientes" de sus SNA, lo que significa que el simulador sólo debe indicarle a esta entidad que evalúe un SNA y cada entidad sabrá qué significa dicho SNA, o sea qué propiedad deberá evaluar o qué función deberá invocar.

Volviendo a la evaluación de expresiones numéricas, además de considerar los distintos SNA de GPSS, el simulador debe ser capaz de identificar nombres de entidades, en cuyo caso deberá obtener el valor asociado a dicha entidad (por lo general, su atributo identificador). Adicionalmente, como es de esperarse, las expresiones pueden incluir números, lo cual también debe ser considerado al momento de evaluarla. Todos estos casos fueron considerados dentro de la jerarquía de clases encabezada por la superclase GenericExpression (Imagen 28).

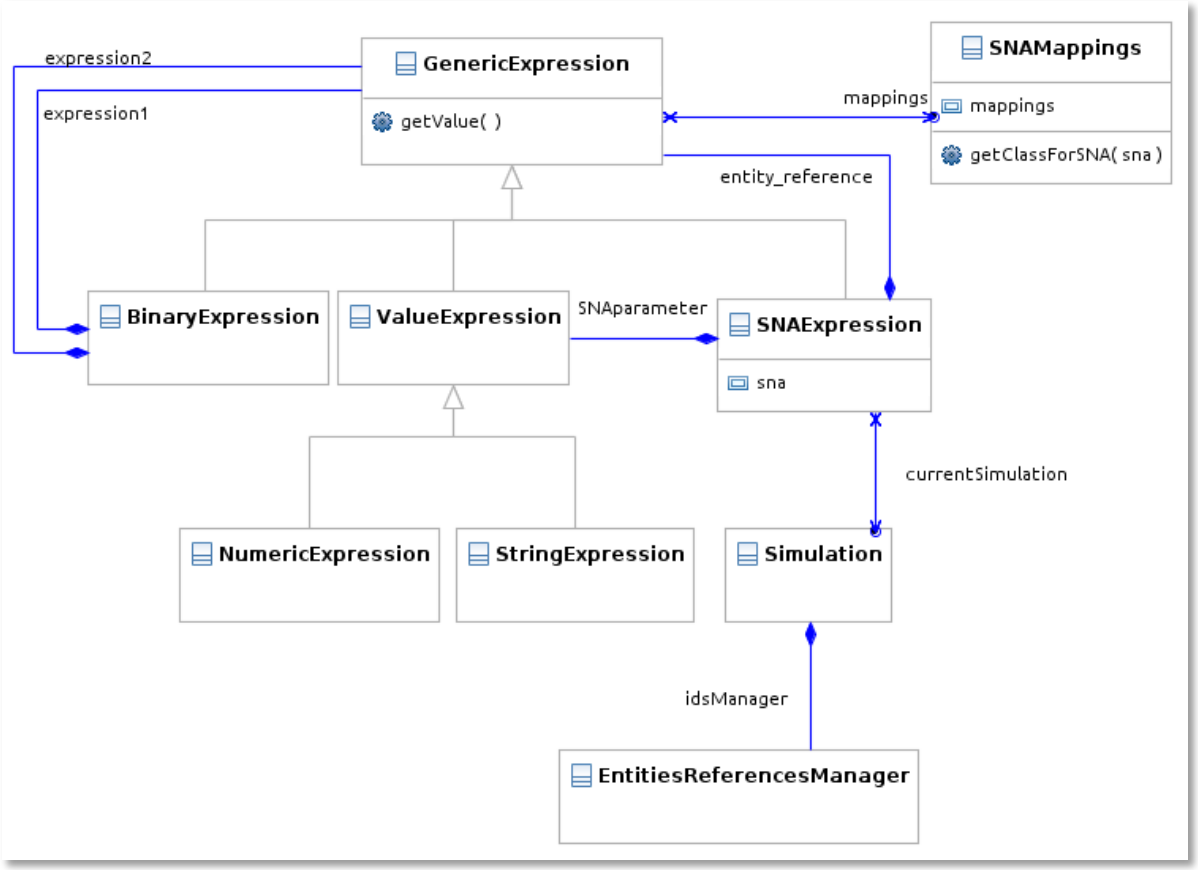


Imagen 28: Principales clases utilizadas para representar y evaluar expresiones numéricas

Interpretación y ejecución de código GPSS

El modelo de entidades presentado en las secciones anteriores por medio de clases, jerarquías de clases y relaciones entre las mismas, permite representar un modelo en GPSS que puede ser ejecutado por el motor de simulación incluido en esta aplicación. Sin embargo, para que las entidades del modelo sean creadas y para que la simulación pueda ser corrida, es necesario introducir de alguna manera las instrucciones de GPSS, en términos de bloques y comandos, que indiquen al simulador qué entidades se utilizarán y cómo interactuarán unas con otras durante la ejecución del modelo. Estas instrucciones son, como es de esperarse, enviadas al motor de simulación en forma de sentencias GPSS. Pero dichas sentencias son sólo cadenas de texto, que deben ser interpretadas por un compilador capaz de reconocer, por un lado, todos los elementos sintácticos del lenguaje, como nombres de bloques y de comandos, atributos y comentarios del programador y, por el otro, de darles un sentido semántico a estos textos, lo que significa identificar las entidades a instanciar y las rutinas que deben

ejecutarse a partir de cada línea de código leída.

La aplicación GPSS PSE funciona de manera similar a otros programas compiladores existentes: recibe como entrada el código fuente de la aplicación en forma de texto plano, y genera como salida un código objeto que puede ser ejecutado, en este caso en términos de clases y objetos del lenguaje Java. Este proceso se realiza en sucesivas etapas, y buena parte del mismo es posible gracias a una herramienta de código abierto conocida como ANTLR (ANother Tool for Language Recognition[13]) [Parr07].

ANTLR es una herramienta generadora de parseadores (*parsers)* libre y de código abierto, que puede ser utilizada para implementar tanto lenguajes específicos de dominio (DSL: *domain-specific languages*) [Mernik05], como lenguajes de programación reales. Esta herramienta, desarrollada en lenguaje Java, fue implementada y es mantenida por el profesor Terence Parr, de la Universidad de San Francisco (Estados Unidos).

ANTLR requiere una gramática libre de contexto como entrada, y genera una herramienta capaz de tomar texto plano, verificar si cumple con dicha gramática y por último generar el código objeto acorde a las reglas descriptas en dicha gramática. Para ello, ANTLR utiliza algunas construcciones auxiliares que se integran en archivos de especificación, gramáticas en lenguaje EBNF (Extended Backus-Naur), una versión ampliada de BNF que permite generar reglas más simples gracias al soporte de bucles dentro de la gramática y de subreglas de las enumeraciones, entre otras ventajas [Aaby96]. Se dice que ANTLR es un *parser LL* [Li95], lo que significa que realiza un análisis desde lo más general a lo más particular (*top-down*), que interpreta el código fuente de izquierda a derecha (*left to right*) y que construye derivaciones por la izquierda (*leftmost*) de cada sentencia. Junto a otros programas similares, ANTLR es incluido comúnmente dentro de la categoría de meta-programas: programas que sirven para crear otros programas.

Entre las principales características de ANTLR, se destaca su capacidad de realizar los tres tipos de análisis requeridos por un compilador (léxico, sintáctico y semántico), así como también de generar el código objeto en caso de indicarse. Para ello, los analizadores construidos con ANTLR trabajan en sucesivas etapas (Imagen 29)

[13] ANTLR, página web: http://www.antlr.org/.

produciendo datos y nuevas estructuras a partir del resultado de la etapa anterior. Estas etapas son:

1. *Análisis léxico*: la primera etapa toma el código fuente como secuencia de caracteres, y lo procesa para así detectar los símbolos o componentes léxicos (*tokens*) que servirán para futuras etapas de análisis. El analizador léxico utiliza las reglas definidas en la gramática EBNF como soporte para detectar estos *tokens*.

2. *Análisis sintáctico*: en la segunda etapa se procesan los *tokens* junto con el código fuente y se generan estructuras conocidas como árboles de sintaxis abstracta (AST: *Abstract Syntax Tree*), que es una estructura simplificada del código fuente. En esta estructura, cada nodo denota una construcción que ocurre en el código fuente.

3. *Análisis semántico*: en la tercera etapa, el analizador utiliza como entrada el árbol sintáctico generado en la etapa previa para comprobar restricciones de tipos y otras limitaciones semánticas y preparar la generación de código objeto. El analizador semántico se compone de un conjunto de rutinas independientes, que suelen hacer uso de una pila (la pila semántica) que contiene la información semántica asociada a los operandos (y en ocasiones a los operadores también) en forma de registros semánticos.

4. *Código objeto*: la última etapa consiste en la generación del código objeto, el cual podrá luego transformarse en código de máquina o código ejecutable. El generador de código toma el código intermedio generado en la etapa anterior, también estructurado dentro de un AST.

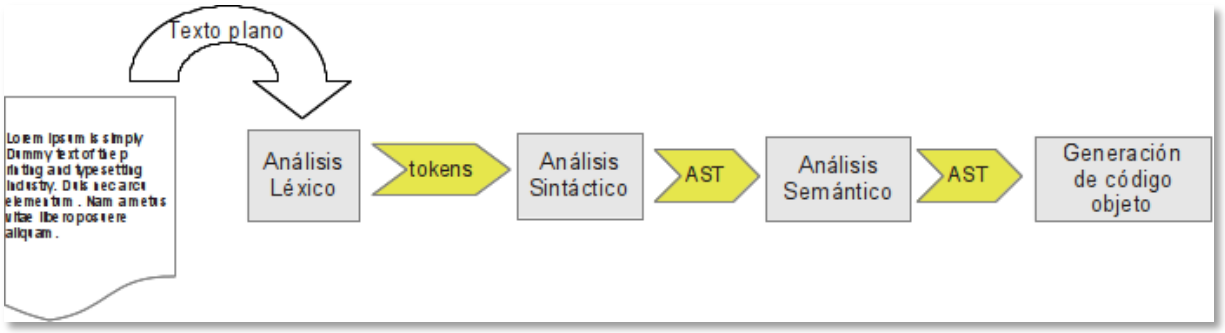


Imagen 29: etapas de análisis de ANTLR

Las facilidades que ofrece ANTLR han sido integradas dentro de GPSS PSE. Gracias a esto, GPSS PSE puede recibir código GPSS en forma de texto plano, analizarlo y generar el modelo GPSS que se corresponde con las definiciones contenidas en el código ingresado. Dentro del modelo de GPSS se incorporó un archivo, denominado GPSS.g, que contiene la gramática de GPSS incluyendo nombres de bloques, de comandos, de SNA, y reglas para construir sentencias GPSS correctas. Esta gramática es utilizada primero por la clase *GPSSLexer*, que es la encargada de realizar el análisis léxico y el análisis sintáctico, así como también de generar y completar la lista de *tokens* a partir del resultado de dichos análisis. El resultado de estos análisis es tomado por la clase *GPSSParser,* que se encargará de realizar el análisis semántico (*parsing*) tomando como dato de entrada la lista de *tokens* generada previamente.

Esta secuencia (Imagen 30) es controlada por la clase *GPSSInterpreter*, que puede recibir como entrada tanto una secuencia de caracteres como un archivo de texto, y genera a partir de allí un objeto de clase *ANTLRInputStream*, el cual es luego utilizado por el método *interpret* para invocar al *lexer*. La clase *GPSSInterpreter* también se encargada de realizar una validación extra sobre los parámetros ingresados en bloques y comandos: una vez que el *lexer* y el *parser* han verificado que no existen errores sintánticos y semánticos, es preciso verificar que ciertas dependencias propias del modelo de GPSS se cumplan. Un claro ejemplo se presenta en las distintas combinaciones de los parámetros del bloque GENERATE: si no se utilizan los parámetros A (tiempo medio entre generación de transacciones) y B (desvío del tiempo medio entre generación), es necesario indicar un valor en el parámetro D (límite máximo de generación). Una vez que las etapas de análisis han concluido y que todos los bloques y comandos han sido validados, el método *interpret* retorna un objeto de clase Simulation, el cual, como es de esperarse, representa a una entidad Simulation lista para ejecutarse invocando al método *simulate(),* lo que inicia la ejecución secuencial de la cola de comandos GPSS definidos en el modelo original.

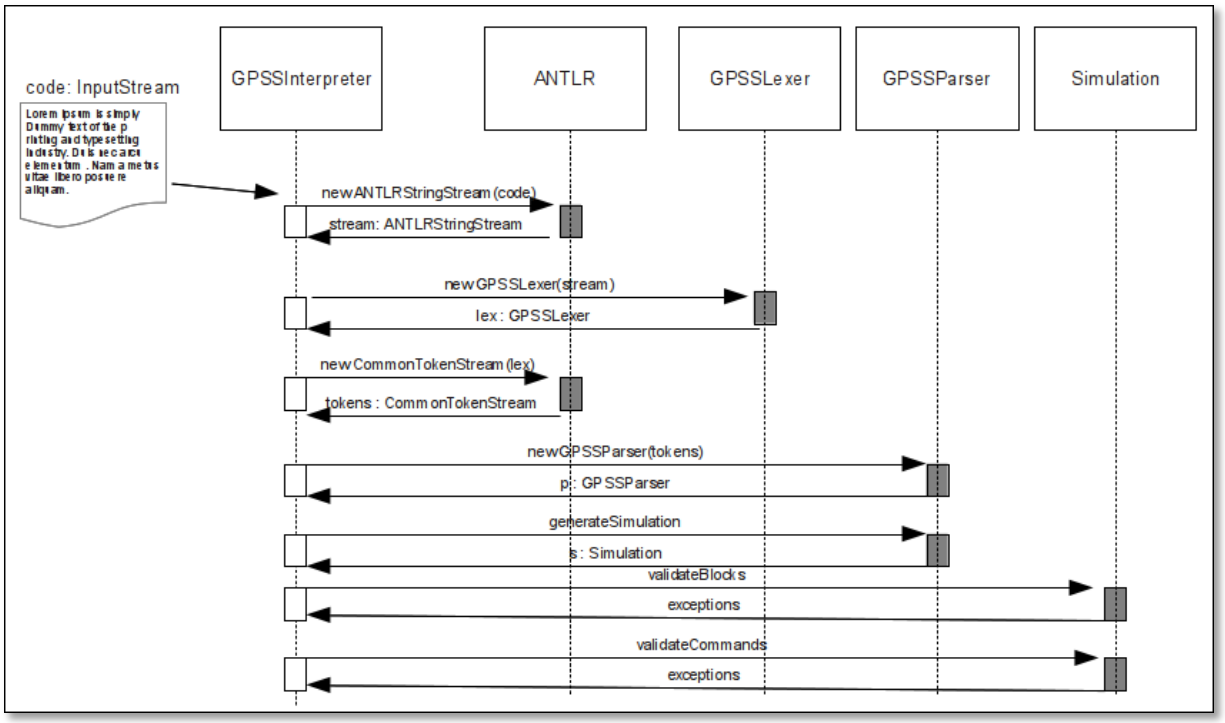


Imagen 30: generación de una simulación ejecutable desde código GPSS

Historial de cambios

El propósito principal detrás del desarrollo de GPSS PSE es el de ayudar a los usuarios a comprender cómo se ha ejecutado la simulación y por qué se han alcanzado los resultados finales. Para lograr esto, es necesario contar con algún mecanismo capaz de mantener un registro de todos los cambios en el modelo que se suceden durante su ejecución, así como también es necesario contar con herramientas para recuperar las distintas entidades en un momento dado de la ejecución.

Como ya se mencionó en el capítulo 3, la ejecución de una simulación es dirigida principalmente por un reloj de simulación que avanza cada vez que la cadena de eventos actuales ha quedado sin eventos, y finaliza cuando el contador de terminación (*termination count*) alcanza el valor cero. Algunas entidades permanentes son creadas antes de comenzar la ejecución, mientras que otras entidades permanentes y todas (o casi todas) las entidades temporales son creadas en tiempo de ejecución. Además, durante la ejecución de la simulación, las entidades transitorias interactúan unas con otras, ingresan a las diferentes cadenas del sistema, y alteran el estado de todas las entidades permanentes. Esto significa que, además de registrar el estado individual de

las diferentes entidades del sistema, el registro de ejecución de simulación debe ser capaz de reflejar las distintas interacciones en cada momento de la simulación.

La existencia de un reloj de simulación que avanza a intervalos irregulares pero discretos brinda una oportunidad para generar un sistema de versionado de la simulación, donde cada nueva versión se corresponderá con cada cambio del reloj de simulación. Si bien esto podría implicar que algunos cambios no se reflejen en el historial (aquellos que han sucedido en un mismo instante de simulación), este mecanismo permite conocer, por un lado, cuáles fueron los intervalos de avance del reloj de simulación y, por el otro, el estado de las entidades y sus interacciones. De todos modos, para el caso particular de aquellas entidades creadas y eliminadas en un mismo instante de simulación, se ha incorporado un mecanismo que permite registrar cuáles son las entidades eliminadas en cada instante del reloj de simulación, asegurando que no se perderá esta valiosa información incluso si no han perdurado por más de un instante de reloj.

A fin de mantener un historial completo de cambios en la simulación, GPSS PSE incluye un módulo que contiene un mecanismo de copiado integral de toda la simulación. Para ello, en cada cambio de reloj, el módulo de copiado parte desde la entidad *Simulation* y se recorre el grafo de composición de entidades presentado previamente, recolectando información de todas las entidades. Asimismo, a medida que recorre el grafo, este módulo debe también registrar las aristas por las que ya ha pasado, para 1) evitar ingresar en un ciclo infinito y 2) registrar también las relaciones entre las entidades, pues cada arista en el grafo se corresponde con una relación entre dos entidades. Por ejemplo, si desde la entidad CEC, perteneciente a la entidad *Simulation*, se llega a la entidad Transaction con identificador 10, quiere decir que la transacción 10 se encontraba en la CEC en ese instante de reloj, o sea que existía una relación directa entre la transacción 10 y la entidad CEC.

El mecanismo arriba descripto genera una fotografía instantánea del grafo de simulación con cada cambio del reloj de simulación. Como ya se mencionó, esta fotografía incluye tanto a las entidades de la simulación como a sus relaciones. Una vez que todo el grafo ha sido copiado, debe ser almacenado en algún medio persistente que asegure que este historial no se perderá, y que permita luego su acceso. Para ello, todas

las entidades y relaciones son almacenadas en una base de datos MySQL[14]. Dado que este es un motor de base de datos relacional, todos los objetos (entidades) son transformados mediante la librería JPOX [Pons10][Gómez12] antes de ser almacenados. Como es de esperarse, este mapeador objeto-relacional es también utilizado al momento de recuperar las entidades, como se verá más adelante.

Si bien el uso en una base de datos resulta muy práctico tanto para almacenar como para recuperar las entidades de la simulación, surge un problema de performance relativo a los tiempos requeridos para almacenar la información en la base de datos. Mientras que la ejecución de un modelo simple en GPSS suele tomar muy pocos segundos, el almacenamiento de toda la información generada en cada cambio del reloj de simulación puede tomar muchos segundos, e incluso minutos si el modelo contiene o genera dinámicamente una gran cantidad de entidades. A fin de evitar que el módulo de persistencia (*PersisterThread*) retrase considerablemente la ejecución de la simulación, fue implementado de manera separada al módulo de copiado, y su ejecución se realiza bajo un hilo paralelo a la simulación. Esto significa que, al iniciar la ejecución de un modelo GPSS, la aplicación GPSS PSE lanza dos hilos de ejecución, uno dedicado a la ejecución en sí, lo cual incluye también el copiado del grafo de entidades, y otro dedicado al almacenamiento de estas entidades. Estos dos hilos se coordinan bajo el modelo de concurrencia conocido como productor-consumidor: cada vez que se genera una nueva copia de la entidad *Simulation*, y desde allí de todo el grafo de la simulación, se almacena en memoria en una cola de "simulaciones listas". Por otro lado, el módulo de persistencia observa permanentemente esta cola, y cada vez que encuentra allí una nueva entidad Simulation, la quita de la cola e inicia el proceso de guardado en la base de datos relacional; esto implica el mapeo de todos los objetos (entidades) y relaciones entre ellos en tablas y relaciones de MySQL, la generación de identificadores únicos de objetos, y la ejecución de las consultas SQL necesarias para persistir esta información. El guardado de las entidades de la simulación se realiza por alcance, con lo cual al guardar la entidad *Simulation* se comenzará un guardado en cascada que alcanzará a todas las entidades alcanzables desde allí.

Todas las entidades *Simulation* almacenadas en la base de datos serán, como ya se

[14] MySQL :: The world's most popular open source database, página web: http://www.mysql.com.

explicó, capturas instantáneas de la simulación en un instante dado. Esto permite considerar a cada captura como una simulación independiente del resto, y estudiarla de manera aislada o como una secuencia de simulaciones que forman una ejecución completa. Para permitir regenerar la secuencia completa, antes de iniciar la ejecución de la simulación se genera un identificador único universal (UUID [Leach05]) y se le asigna a la entidad Simulation. Este identificador se mantiene sin alterar durante toda la ejecución, y se copia también en cada copiado del grafo de simulación. Gracias a este identificador, es posible regenerar una ejecución completa de un modelo, tomando todas las entidades *Simulation* que contengan el mismo UUID y ordenándolas según el valor del reloj de simulación.

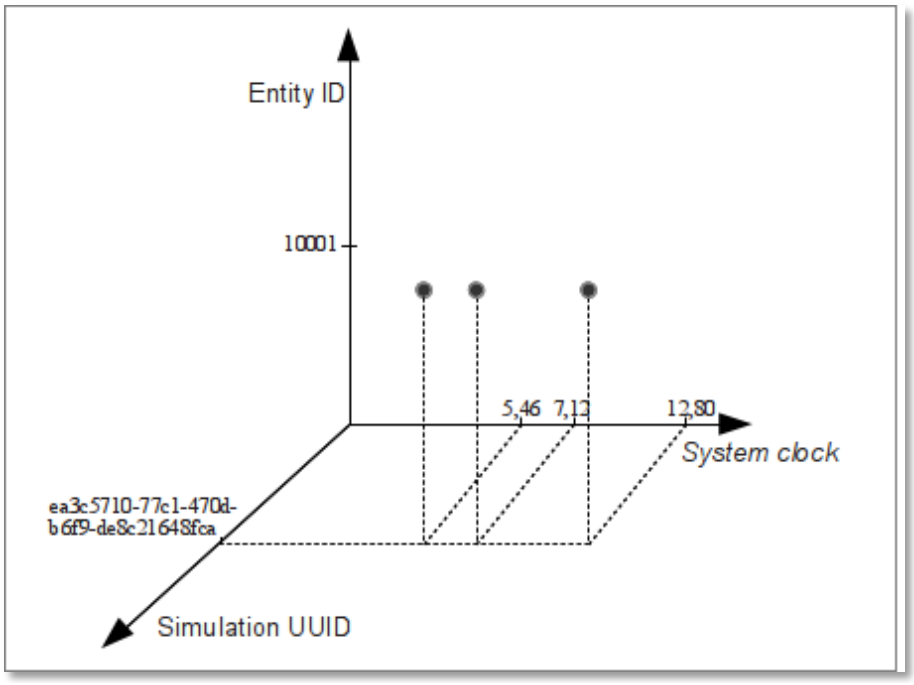


Imagen 31: Versionado de la entidad 10001 en distintos tiempos de reloj para una misma simulación

El uso de este identificador también es un elemento clave a la hora de recuperar una entidad en particular. GPSS utiliza los mismos identificadores para las entidades permanentes en distintas simulaciones; por ejemplo, asigna el ID 10000 a la primera facilidad, el 10001 a la primera cola, y así sucesivamente. Esto permite identificar a una entidad unívocamente en una corrida de simulación particular, pero no permite distinguir dicha entidad de diferentes corridas, dado que siempre tendrá el mismo ID. Sin embargo, el uso del UUID para identificar unívocamente a la ejecución de una

simulación, junto a la integración de todas las entidades al grafo de simulación, permite también identificar unívocamente a cualquier entidad más allá de la simulación donde se generó. Asimismo, dado que las diferentes capturas de una misma corrida son identificadas por el valor del reloj de simulación en cada captura, es posible identificar una captura en un instante de reloj particular de una entidad particular en una simulación particular. Del mismo modo, es posible recuperar todo el historial de una entidad en particular, a partir de la recuperación de todas las capturas de dicha entidad en un mismo grafo de simulación, o sea alcanzable desde todas las entidades *Simulation* bajo un mismo UUID. Esto significa que para acceder a toda la información de una entidad particular de la base de datos, es preciso especificar el ID de la entidad (entity ID) en la simulación, y el UUID de la simulación; y si además se desea recuperar información de dicha entidad en un instante de simulación, deberá especificarse el valor del reloj de simulación para la entidad *Simulation* bajo el mismo UUID (Imagen 31).

Interacción entre el Entorno Interactivo y GPSS PSE

Hasta aquí se han descrito dos herramientas de software diseñadas para ayudar a comprender un software de simulación a eventos discretos, conocido como GPSS, pero que atacan el problema desde dos ángulos muy diferentes. El Entorno Interactivo de GPSS se sirve para construir modelos GPSS válidos de manera interactiva, con lo cual busca ayudar durante la etapa de aprendizaje del lenguaje y sus entidades; por otro lado, el motor de simulación con persistencia (GPSS PSE) permite ejecutar modelos escritos en lenguaje GPSS mientras registra los cambios que suceden en el modelo, y tiene como objetivo ayudar a comprender cómo los modelos se han ejecutado, o sea, cómo funciona el simulador por dentro. Estas dos aplicaciones funcionan de manera independiente entre sí. El Entorno Interactivo puede ejecutarse incluso sin el uso de un servidor web, y utilizarse para crear modelos válidos o simplemente como soporte de la documentación de GPSS, ya que ofrece una interfaz muy simple con información sobre bloques, comandos, operadores, parámetros y entidades del modelo. Por el otro lado, GPSS PSE puede utilizarse como una herramienta de ejecución de transacciones, y funciona incluso como un *plugin* de Eclipse, ofreciendo al usuario un área de texto para introducir código GPSS, y una interfaz muy simple que muestra información sobre la ejecución de la

simulación.

En el contexto de este trabajo, es necesario presentar estas dos aplicaciones como una única aplicación integrada, de manera similar al funcionamiento de los entornos de desarrollo integrados (IDE), ya conocidos por los alumnos. Los usuarios de la aplicación deben ser capaces de crear modelos simples de manera interactiva; una vez construidos, los modelos deben poder ejecutarse directamente desde la misma aplicación y, más importante, los usuarios deben poder visualizar los resultados de la ejecución de su modelo, lo cual significa en este contexto estudiar el avance de la simulación junto a todas las entidades a lo largo del tiempo de simulación. Para que esta integración funcione de manera transparente, se desarrolló una aplicación que brinda las funciones básicas para que las otras dos aplicaciones se comuniquen. Esta aplicación fue desarrollada en lenguaje Java, y construida sobre el *framework* Spring MVC[15]. Al igual que la mayoría de las aplicaciones web construidas con este *framework*, esta aplicación se ejecuta dentro de un contenedor web como Apache Tomcat[16] o Jetty[17], y su diseño es muy simple. Existe una clase, llamada *GPSSController*, que se encarga de transformar (mapear) requerimientos HTTP en métodos Java, y de transformar los objetos Java en JSON[18], para luego enviarlos al navegador web, proceso que se conoce como serialización de objetos [Hericko03].

Desde el punto de vista del uso, la aplicación web es invocada por los usuarios en 3 momentos puntuales: al cargar la interfaz de trabajo, al ejecutar la simulación, y al explorar los resultados de la ejecución. A continuación, se explica brevemente el rol que cumple esta aplicación en cada caso:

a) *Carga inicial*: como ya se mencionó, el Entorno Interactivo funciona perfectamente sin un servidor web o un servidor de aplicaciones detrás. Esto es posible porque la lógica de carga de librerías y módulos se encuentra embebida dentro de la misma aplicación, mediante la librería RequireJS. Gracias a esto, al recibir una petición (*request*) a la URL *index*, la aplicación web sólo debe generar y enviar el archivo index.jspx, cuyo único contenido

[15] Spring Framework, página web: http://www.springsource.org/spring-framework.
[16] Apache Tomcat, página web: http://tomcat.apache.org/.
[17] Jetty Servlet Engine and HTTP Server, página web: http://tomcat.apache.org/.
[18] JSON Javascript Object Notation, página web: http://www.json.org/.

será el esqueleto HTML vacío correspondiente a la interfaz de usuario, y el módulo RequireJS que se encargará de cargar el resto de los archivos. Luego, durante la construcción del modelo GPSS, toda la acción sucederá del lado del cliente, y no se requerirá acceder al servidor web salvo para recuperar algún archivo javascript por primera vez.

b) *Ejecución de la simulación*: cuando el usuario decide ejecutar su modelo, se generará desde el Entorno Interactivo el código GPSS de manera automática, el cual será luego enviado en forma de texto plano mediante AJAX [Garrett05] al servidor de aplicaciones. Aquí, la aplicación web tomará este texto e invocará a la aplicación GPSS PSE para que se encargue de ejecutarlo. Para ello, como ya se explicó, creará un nuevo objeto de la clase *GPSSInterpreter*, al cual le enviará el código GPSS y del cual recibirá como resultado (en caso de que no haya habido errores durante la ejecución de la simulación) un objeto de clase *Simulation*. A partir de este objeto, se extraerá el UUID que identifica esta ejecución, el cual luego será enviado al cliente como respuesta a la solicitud de ejecución del modelo. Finalmente, el cliente web almacena el valor de este UUID, para luego utilizarlo para recuperar información sobre la ejecución de esta simulación.

c) *Exploración de resultados*: la cantidad de información que se genera al ejecutar una simulación puede sobrecargar fácilmente al navegador web del usuario si se envía toda junta para su procesamiento. Por ello, se implementó un mecanismo de exploración parcial de resultados, en el cual sólo se envía al navegador web aquella información que éste mostrará al usuario, y solamente eso. Por ejemplo, al finalizar la ejecución, se le presenta al usuario un listado de tiempos de simulación, que se corresponden con todos los valores que ha tomado el reloj de simulación durante la ejecución del modelo. Para conocer el estado de la simulación en cada instante del reloj, el usuario debe seleccionar el valor deseado. Esto provoca el envío de un requerimiento hacia el servidor, que contendrá el valor del reloj de simulación y el UUID de la simulación, enviado antes como respuesta de la finalización de la ejecución. Con esta información, la aplicación web le solicitará a la aplicación GPSS PSE

que recupere la fotografía instantánea de la simulación con ese UUID y en ese instante de reloj. El grafo de simulación completo es recuperado por GPSS PSE y posteriormente transformado al formato JSON por la aplicación web. Finalmente, este objeto JSON es enviado como respuesta a la petición original enviada por el cliente web, el cual lo procesará y mostrará su información de manera comprensible para el usuario.

Diseño orientado a la enseñanza

En la sección anterior se introdujeron varios aspectos relativos al diseño de GPSS Interactivo y sus dos principales aplicaciones, el Entorno Interactivo cuyo propósito es asistir a los usuarios al momento de crear modelos en GPSS, y el motor de simulación GPSS PSE, cuyo objetivo es ejecutar modelos GPSS mientras mantiene un historial del avance de la simulación. Desde el punto de vista del diseño pedagógico, estas aplicaciones tienen como objetivo principal fomentar el desarrollo de determinados procesos cognitivos que ayudan a los alumnos a fijar mejor los conceptos subyacentes propios del lenguaje GPSS y del motor de simulación:

- contribuir a generar la observación minuciosa,
- activar la evocación,
- poner en juego la intuición y el pensamiento creativo,
- favorecer la comprensión y la interpretación,
- ayudar a los procesos de relación,
- ayudar al proceso de análisis y especialmente al proceso de síntesis,
- propiciar el razonamiento (deductivo, inductivo y crítico),
- activar la imaginación.

El diseño del curso de Modelos y Simulación debe ser pensado con el objetivo de aprovechar las características de este desarrollo, mediante la organización y distribución de los conceptos a lo largo de la cursada, y la distinción de temáticas principales y secundarias. El uso de este desarrollo para el aprendizaje, sumado al correcto

acompañamiento docente durante el dictado del curso, reduce de manera considerable el aprendizaje por recepción mientras fomenta el aprendizaje por descubrimiento. Como se verá más adelante, la combinación entre esta herramienta, una planificación adecuada y una metodología de organización y dictado de los cursos permite también optimizar el tiempo dedicado al aprendizaje de los conceptos básicos, lo cual permite estudiar otros temas más a fondo y lograr así un aprendizaje aún más significativo.

En esta sección, se detallarán las características pedagógicas con las que ambas aplicaciones fueron concebidas, y qué aporte realiza cada una de estas características al proceso de enseñanza del simulador GPSS.

Entorno Interactivo

Una de las principales características de GPSS es la gran variedad de entidades con las que los desarrolladores cuentan al momento de construir sus modelos. Las entidades de GPSS fueron concebidas con propósitos diferentes y, al momento de utilizarlas, los desarrolladores deben conocer con certeza qué funciones y rutinas posee cada una, y qué herramientas de análisis brindan al finalizar la simulación. Esto les permitirá no sólo realizar modelos correctos, sino también aprovechar al máximo el potencial del lenguaje y sus entidades.

Las entidades de GPSS no son accedidas directamente por los desarrolladores. Por el contrario, GPSS se vale de un amplio conjunto de sentencias de dos tipos, comandos y bloques, que permiten al desarrollador aprovechar las distintas entidades. Los comandos sirven para definir ciertos parámetros de algunas de estas entidades antes de comenzar la simulación, como, por ejemplo, la cantidad de unidades de un *storage*, el valor inicial de un *savevalue,* o el argumento, tipo y valores de una *function*. Algunos comandos poseen efectos secundarios; por ejemplo, el comando *START* inicializa el valor del atributo *termination count* de la simulación, y a su vez provoca el inicio de la ejecución de las transacciones creadas a partir de los bloques *GENERATE*. Los bloques, por otro lado, sirven para invocar las subrutinas internas de las distintas entidades. La principal diferencia entre bloques y comandos radica en el momento de ejecución de cada uno, o más precisamente, en el responsable de la ejecución de cada uno. Mientras que los comandos son ejecutados automáticamente por el motor de simulación uno

detrás de otro, los bloques son accedidos y ejecutados por las transacciones, y su orden de ejecución no es necesariamente secuencial, dado que las transacciones pueden realizar saltos entre bloques, ingresar en bucles o incluso detenerse a la espera de algún evento.

Uno de los inconvenientes que se presentan al momento de aprender GPSS es la diferenciación entre qué es un bloque y qué es un comando. Probablemente, esto se deba a que la forma de escribir estas sentencias es prácticamente igual, y además el compilador de GPSS permite que el usuario combine bloques y comandos en el modelo. Esto es posible porque, al momento de analizar el código fuente, el compilador separa los comandos por un lado, para organizarlos en una cola según el orden de aparición, y los bloques por el otro, para generar la lista de bloques por la que las transacciones avanzarán. Esta característica resulta muy práctica para el desarrollador experimentado en el lenguaje GPSS, ya que posibilita que los comandos que definen entidades complejas se coloquen junto a los bloque que las utilizan. Por ejemplo, al definir una entidad TABLE —comparable con un histograma—, el desarrollador debe utilizar el comando homónimo y asignarle un nombre, un atributo a "tabular" (marcar dentro de una tabla), el tamaño las clases de equivalencia, la cantidad de clases de equivalencia, y el límite superior de la primera clase de equivalencia. Cuando el modelo tiene pocas entidades, es fácil recordar los parámetros de creación de las mismas. Pero, en caso de contar con muchas entidades, por ejemplo, una docena de entidades TABLE, resulta considerablemente más difícil recordar con qué valores se definió cada una. Gracias a la posibilidad de combinar bloques y comandos, al momento de insertar el comando TABULATE, el programador puede contar con toda la información de creación de la tabla y así evitar confusiones.

Si bien la combinación de bloques y comandos puede ayudar al programador experto, puede generar confusión en los alumnos que están aprendiendo a utilizar este lenguaje. Es por esto que el Entorno Interactivo fue diseñado con el fin de marcar fuertemente esta diferencia, tanto en las sentencias disponibles para utilizar como también en el código GPSS ya insertado por el desarrollador. Para ello, la pantalla principal de trabajo posee ciertas áreas donde se presenta sólo información relativa a los comandos y otras áreas donde sólo se presenta información sobre los bloques. Estas áreas son (Imagen

32):

1. *Cola de comandos*: se muestra la cola de comandos incorporados por el programador. En esta área no se muestran los nombres de las entidades ni los parámetros de creación, pues el objetivo es destacar los comandos utilizados y el orden de los mismos.
2. *Lista de bloques*: de manera similar a la cola de comandos, en esta área se muestran los bloques que el programador ha utilizado, haciendo énfasis en el orden del mismo. No se muestran aquí etiquetas ni campos de los bloques.
3. *Código de comandos*: aquí sí se genera todo el código de los comandos insertados, en el mismo orden que se muestran en la cola de comandos. Este código incluye nombre de entidades, número de línea, y los parámetros del comando.
4. *Código de los bloques*: en este espacio se genera todo el código de la lista de bloques. Esto incluye el número de línea o la etiqueta del bloque en caso de que se le haya asignado alguna, el nombre del bloque, y la lista de parámetros utilizados en cada bloque. A diferencia de lo que sucede en el lenguaje GPSS estándar, aquí se muestran también los parámetros no utilizados como espacios vacíos. El objetivo aquí es destacar la existencia de otros parámetros disponibles para el bloque, que no están siendo utilizados pero que están allí en caso de ser necesarios.
5. *Bloques disponibles*: en este espacio el programador puede acceder a la lista completa de bloques que puede incorporar a sus modelos.
6. *Comandos disponibles*: al igual que el espacio anterior, aquí se muestran los comandos disponibles para el programador.

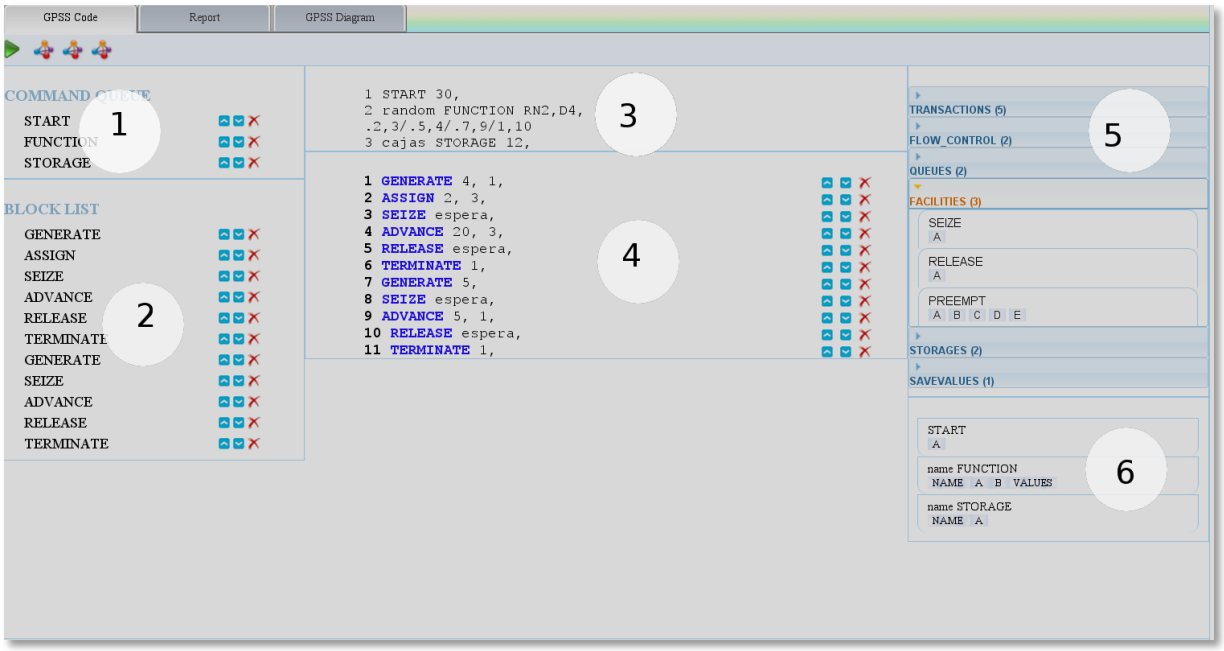


Imagen 32: Principales áreas de la interfaz de trabajo de GPSS Interactivo

Si bien el objetivo de esta herramienta no es desarrollar modelos extensos sino aprender a usar los elementos del lenguaje, es posible que durante la etapa de aprendizaje, algunos modelos se vuelvan lo suficientemente amplios como para generar cierta confusión en las correspondencias de elementos de las áreas 1 y 3 (cola de comandos con código de comandos), y de las áreas 2 y 4 (lista de bloques con código de bloques). Esto se debe a que, a medida que el código se expande, será más difícil identificar a qué elemento de las áreas a la izquierda se corresponde cada línea de código del área central. Es por esto que se incorporó un mecanismo de resaltado de sentencias (Imagen 33), mediante el cual, al pasar el ratón por encima de cualquier sentencia en cualquiera de las 4 áreas que representan el modelo dinámico creado por el alumno, se destacará el elemento par en el área par correspondiente.

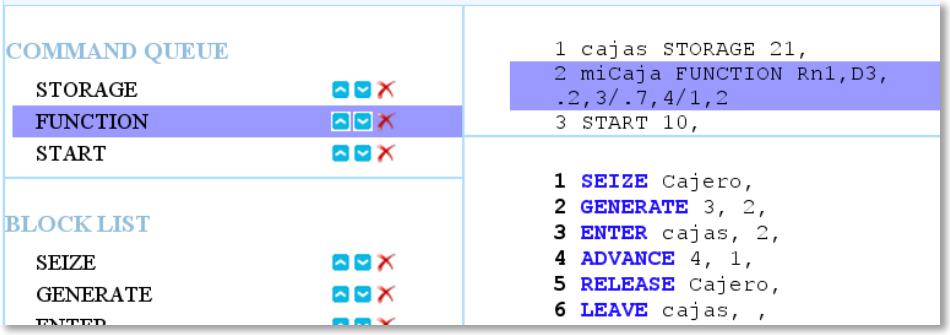


Imagen 33: Resaltado de elementos entre las distintas áreas

Otra de las características de esta aplicación es el resaltado de la sintaxis GPSS. En el espacio dedicado al código de bloques (área 4) es posible identificar: etiquetas o números de línea, en color negro resaltado, nombres de bloques en color azul resaltado, y parámetros de los bloques en forma de texto común. De este modo, se busca que el usuario distinga más fácilmente los términos propios del lenguaje GPSS del resto del código GPSS ingresado.

La organización secuencial de la cola de comandos y de la lista de bloques no es un tema menor en este lenguaje. Es importante aquí que los alumnos comprendan que cada sentencia —ya sea bloque o comando— funciona como una unidad de ejecución, de manera similar a lo que sucede con una subrutina en los lenguajes de programación tradicionales. Y que la posición de estas unidades de ejecución, respecto al resto de las unidades de su mismo tipo, determinará la manera en que su modelo se ejecutará. Por ello, se incorporó en este Entorno Interactivo la posibilidad de mover bloques y comandos a través del modelo. Esto puede realizarse por medio de unas pequeñas flechas de color celeste (hacia arriba y hacia abajo) localizadas junto a cada bloque y comando, o también mediante el uso del ratón, arrastrando y soltando las sentencias. Aquí, se busca nuevamente destacar la diferencia entre los bloques y los comandos; por ello, al mover sentencias de un tipo dentro de un área, no se permite que se combinen con sentencias de otro tipo, o lo que es lo mismo, que las sentencias que están siendo movidas sean enviadas a otra área (Imagen 34). Por ejemplo, al arrastrar un bloque, el alumno nunca podrá soltarlo en el espacio de comandos, sólo podrá hacerlo dentro del mismo espacio de bloques donde se encontraba el bloque en cuestión.

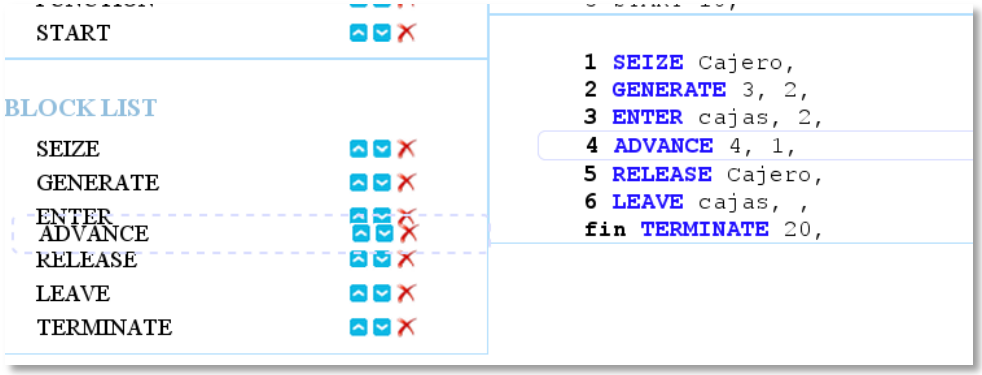


Imagen 34: Arrastrar y soltar (*drag&drop*) bloques y comandos

Construcción asistida de modelos

El Entorno Interactivo fue creado con el objetivo de asistir a los alumnos durante la construcción de sus modelos. Durante esta etapa, es necesario que los alumnos conozcan con qué herramientas cuentan, para qué sirven las mismas, cómo se relacionan con las entidades de GPSS, y cómo pueden utilizarlas. Es por esto que el Entorno Interactivo cuenta con un conjunto de herramientas que atacan estos problemas y que fomentan el aprendizaje del lenguaje GPSS.

Previsualización de sentencias

Como ya se explicó, a la derecha de la pantalla existen dos paneles que muestran al usuario todos los bloques que puede utilizar, y debajo de éstos los comandos disponibles. Al mostrar estas sentencias disponibles, el usuario puede ver además del nombre, una vista previa de los parámetros que requerirá para utilizar dicha sentencia (Imagen 35, Imagen 36). Esta previsualización tiene como principal objetivo ayudar al alumno a realizar una primera identificación de cada bloque o comando, a partir de la cantidad de parámetros y de las características de los mismos.

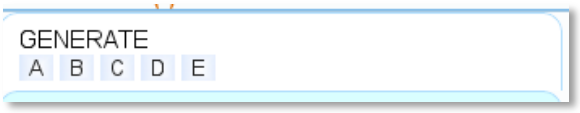


Imagen 35: Vista previa del bloque GENERATE

Imagen 36: Vista previa del comando FUNCTION

Agrupamiento de bloques

Se han creado también varias subcategorías para clasificar los distintos tipos de bloques. Estas categorías se corresponden con las entidades con las que los bloques interactúan en caso de existir (por ejemplo Facilidades), o con la función principal que los mismos realizan (por ejemplo Control de flujo). El objetivo aquí es afianzar la relación que algunos bloques poseen con algunas entidades o su uso típico (Imagen 37). Algunos grupos de bloques surgen de manera natural, como por ejemplo el grupo de bloques de facilidades, con los bloques SEIZE, RELEASE y PREEMPT, o el grupo de los almacenes, con los bloques ENTER y LEAVE. Sin embargo, la relación de otros bloques

con sus entidades no es tan directa en un primer análisis, en especial cuando se está comenzando a aprender las entidades. Por ejemplo, el grupo de bloques de transacciones posee los bloques GENERATE y TERMINATE, que sirven para crear y terminar transacciones, el bloque ADVANCE que sirve para posicionar transacciones en la FEC, y el bloque BUFFER que se utiliza para colocar la transacción al final de la CEC.

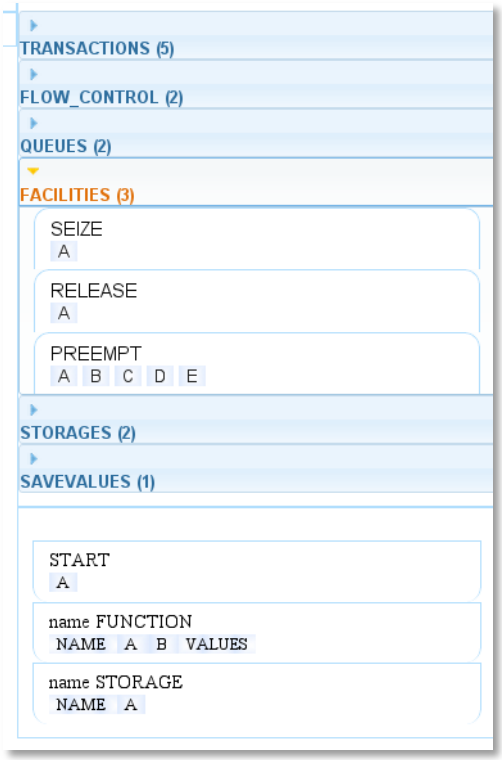


Imagen 37: Bloques agrupados según la entidad con la que trabajan

Programación interactiva mediante diálogos

Al seleccionar cualquiera de estas sentencias, se desplegará delante de la pantalla un diálogo que presentará al usuario toda la información necesaria para utilizar la sentencia elegida. Los diálogos son el único mecanismo existente aquí para insertar código GPSS en el modelo, ya que no se permite la edición directa del código fuente desde aquí. Cada diálogo presenta uno a uno los parámetros que pueden utilizarse para configurar la sentencia. Además, en los casos de los bloques, el usuario puede insertar una etiqueta al bloque (*label*), que luego podrá utilizar para referenciar el bloque desde otros bloques (por ejemplo, para realizar saltos mediante bloques *TEST* o *GATE*). Para cada parámetro, el diálogo muestra un campo de texto en el cual el usuario deberá cargar el valor que

llevará dicho parámetro.

Para cada parámetro se muestra el nombre del mismo junto a una breve descripción de qué podrá especificar allí el usuario. Estos textos se han mantenido en inglés, al igual que en el resto de la aplicación, y se utiliza la misma terminología de los libros y manuales de referencia de GPSS (Imagen 38, Imagen 39). El objetivo aquí es familiarizar al alumno con la interpretación de la documentación de GPSS, requisito fundamental para aprovechar la totalidad del lenguaje, con todas sus entidades y sentencias. Es importante recordar que esta aplicación no tiene como objetivo que el usuario aprenda todo el lenguaje GPSS, sino que aprenda a pensar a la manera de GPSS y que sea capaz de interpretar por sí solo la lógica interna de las entidades y la forma correcta de aprovechar sus sentencias, lo cual le permite ampliar su conocimiento del lenguaje de manera más rápida, y hace más significativo todo lo aprendido.

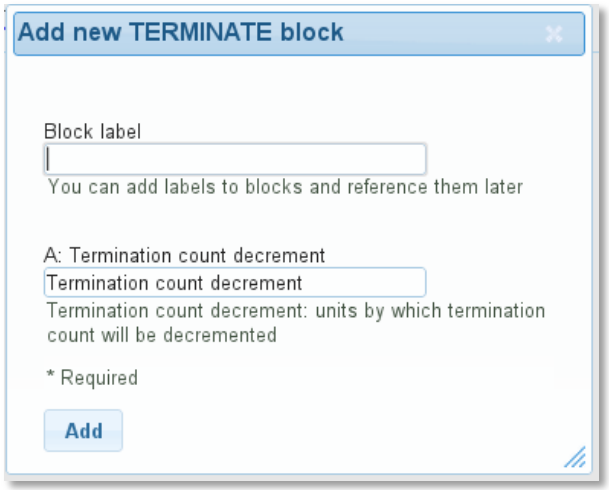


Imagen 38: Diálogo de inserción del bloque TERMINATE

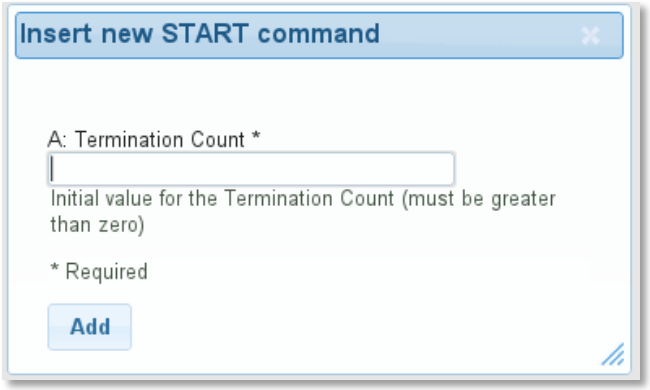


Imagen 39: Diálogo de inserción del comando START

Ordenamiento de parámetros y propuesta de operandos

El orden de aparición de los parámetros en los diálogos es el mismo que se mantendrá al insertar el código de la sentencia, lo cual ayudará al usuario a recordar más fácilmente los parámetros de los bloques (en especial, de los bloques más utilizados, como *GENERATE*). Lo mismo se aplica a los operadores. Mientras que la mayoría de los bloques no requieren operadores, en algunos casos requieren que el programador utilice un operador relacional como mayor (G), mayor o igual (GE), igual (E), menor (L), o menor o igual (LE), y otros bloques requieren un operador condicional como almacén lleno (SF) o facilidad disponible (FV). Algunos problemas comunes que se presentan aquí incluyen el recordar cuáles bloques requieren operadores y cuáles no, el conocer qué tipo de operador se requiere en cada caso, y en el caso de los operadores condicionales, el conocer cuáles están disponibles (la lista completa incluye los operadores FNV, FV, I, LS, LR, M, NI, NM, NU, SE, SF, SNE, SNF, SNV, SV, o U). Los diálogos de bloques que requieren operadores incluyen un campo especial con un selector de operador, el cual sólo contiene los operadores válidos para ese bloque (Imagen 40, Imagen 41). Esto evita que se utilicen operadores inválidos, y recuerda al programador todos los operadores de los que dispone.

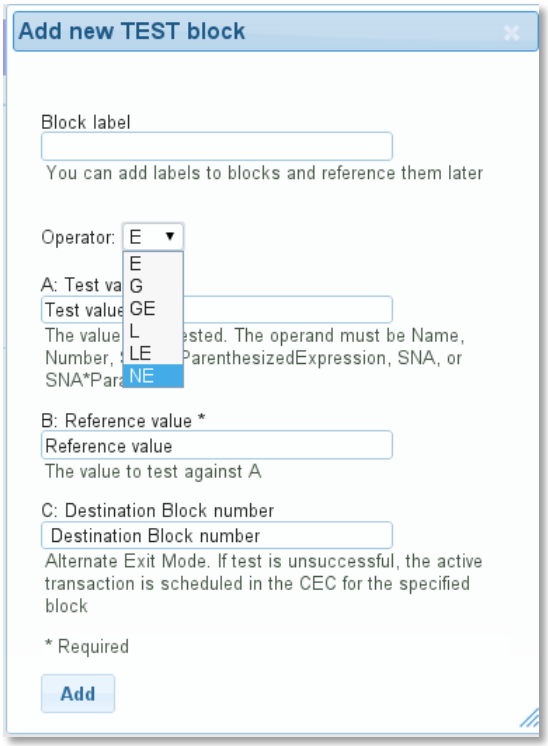


Imagen 40: Operadores relacionales disponibles para el bloque TEST

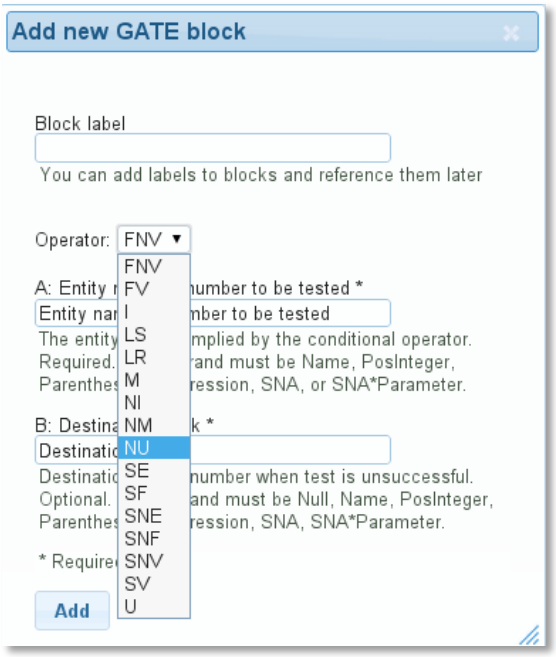


Imagen 41: Operadores condicionales disponibles para el bloque GATE

Parámetros opcionales y parámetros obligatorios

La mayoría de los parámetros de los bloques son opcionales, pero existen algunos parámetros que deben ser necesariamente especificados para que la rutina interna del bloque pueda ser ejecutada. Algunos casos típicos incluyen el nombre del almacén en un bloque ENTER, o el nombre de la cola en un bloque QUEUE. En los distintos diálogos se han destacado con un asterisco (*) aquellos parámetros que son obligatorios, diferenciándolos así del resto de los parámetros. Más allá de esta marca, al momento de confirmar el diálogo, la aplicación verifica que todos los parámetros obligatorios hayan sido completados. De no ser así, el código GPSS no se generará, el diálogo no se cerrará, y se resaltará en color rojo el campo que falta completar, pues el objetivo es minimizar la generación de código erróneo, y destacar la importancia de ciertos parámetros de los bloques. Cada error solucionado se resaltará en color amarillo, destacando así que el problema ya no se encuentra en este parámetro (Imagen 42).

Sugerencia de entidades

Algunos parámetros de determinados bloques requieren que se indique específicamente una entidad, la cual deberá ser del tipo apropiado para que el código sea

correcto. Por ejemplo, el comando PREEMPT espera como primer parámetro el nombre o identificador de una facilidad.

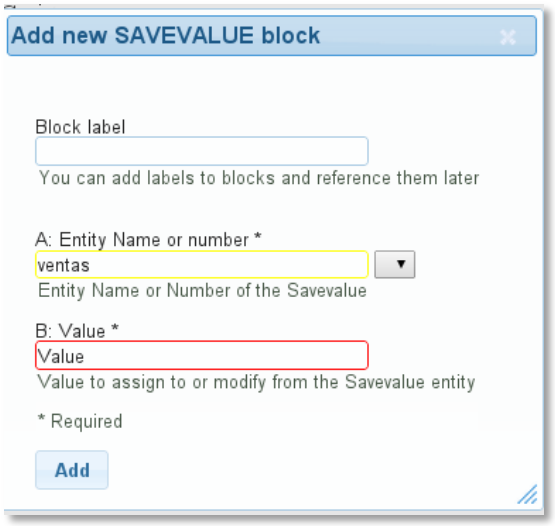


Imagen 42: Parámetros correctos e incorrectos

A diferencia de los lenguajes de programación tradicionales, GPSS no requiere que las entidades sean declaradas antes de ser utilizadas. Esto simplifica mucho los modelos, pero es muy propenso a generar errores pues, cualquier error de escritura generará en el mejor de los casos un error visible, y en el peor no se generará ningún tipo de error y será responsabilidad del usuario detectar y corregir el error. La siguiente secuencia de código ejemplifica esta situación:

```
...
1 SEIZE caja
2 ADVANCE RN1
3 RELEASE caja
...
4 PREEMPT cajas
...
```

El bloque PREEMPT de la línea 4 hace referencia a una facilidad diferente a la utilizada en los bloques SEIZE (línea 1) y RELEASE (línea 3). El compilador de GPSS

asumirá en este caso que se trata de una nueva facilidad, llamada *cajas*, diferente a la facilidad *caja*. Esto presenta muchas veces un problema mayor cuando se está comenzando a aprender este lenguaje, y que hace que detectar errores de código pueda llevar mucho tiempo.

El uso de gran cantidad y variedad de entidades en un modelo permite representar sistemas muy complejos, pero puede también representar un importante desafío para la memoria del programador. Esto se debe a que los entornos de desarrollo tradicionales no poseen mecanismos de asistencia de código, y por lo tanto dependerá de la memoria de cada programador recordar el nombre y tipo de cada una de las entidades de su modelo. Esto se hace aún más problemático si se le suma que el programador está en realidad dando sus primeros pasos en este lenguaje, con lo cual además de recordar el nombre de sus entidades, deberá recordar el nombre de los bloques, los parámetros, el tipo de la entidad, y el comportamiento por defecto de las entidades de ese tipo. Para atacar estos dos problemas, y principalmente poner el foco del aprendizaje en el lenguaje GPSS y sus entidades, y evitar que el alumno se concentre en errores de sintaxis o en recordar todas las entidades que ha creado, el Entorno Interactivo registra todas las entidades utilizadas por el usuario, y se las propone al momento de insertar un bloque (Imagen 43). La aplicación se encarga también de asegurarse que el tipo de la entidad coincida con el tipo que espera el bloque, evitando así la confusión entre entidades de distintos tipos.

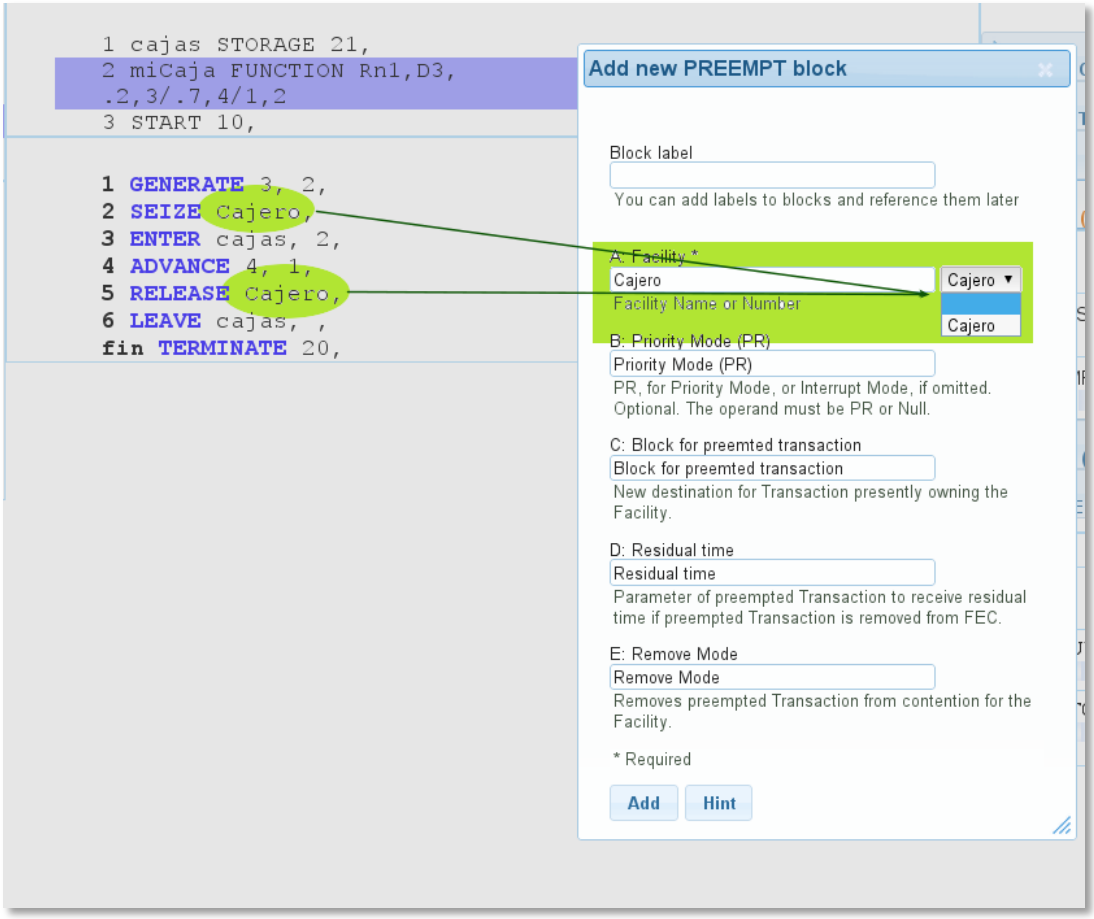


Imagen 43: Sugerencia de entidades estáticas

Imágenes para recordar las entidades

Como ya se ha mencionado, muchos bloques interactúan directamente con entidades permanentes de GPSS, como colas, facilidades o almacenes. Para utilizar apropiadamente los bloques, los alumnos deben primero conocer para qué sirve cada entidad, o qué puede representarse con las distintas entidades. Durante el curso de GPSS, se utilizan constantemente ejemplos típicos de las primeras entidades, pero hasta que estos conceptos sean incorporados, es necesario remarcar constantemente el paralelismo existente entre algunas entidades de GPSS y varios objetos del mundo real. Por ejemplo, la entidad Facility puede asociarse con una CPU con capacidad para ejecutar un proceso a la vez, o con un peluquero capaz de atender un cliente por vez (el ejemplo del peluquero o *barber*, también conocido como "el peluquero dormilón", es un ejemplo típico tanto de GPSS como de cualquier curso de programación concurrente). Para reforzar este punto, se incorporaron galerías de imágenes en aquellos diálogos que se

corresponden con bloques relacionados a dichas entidades. Las imágenes permanecen ocultas en los bloques, y pueden hacerse visibles bajo el botón *Hint* (Ayuda). La galería reproducirá automáticamente la secuencia de imágenes, y para cada imagen se genera una breve descripción que ayuda al usuario a comprender por qué esa imagen se encuentra allí (Imagen 44, Imagen 45).

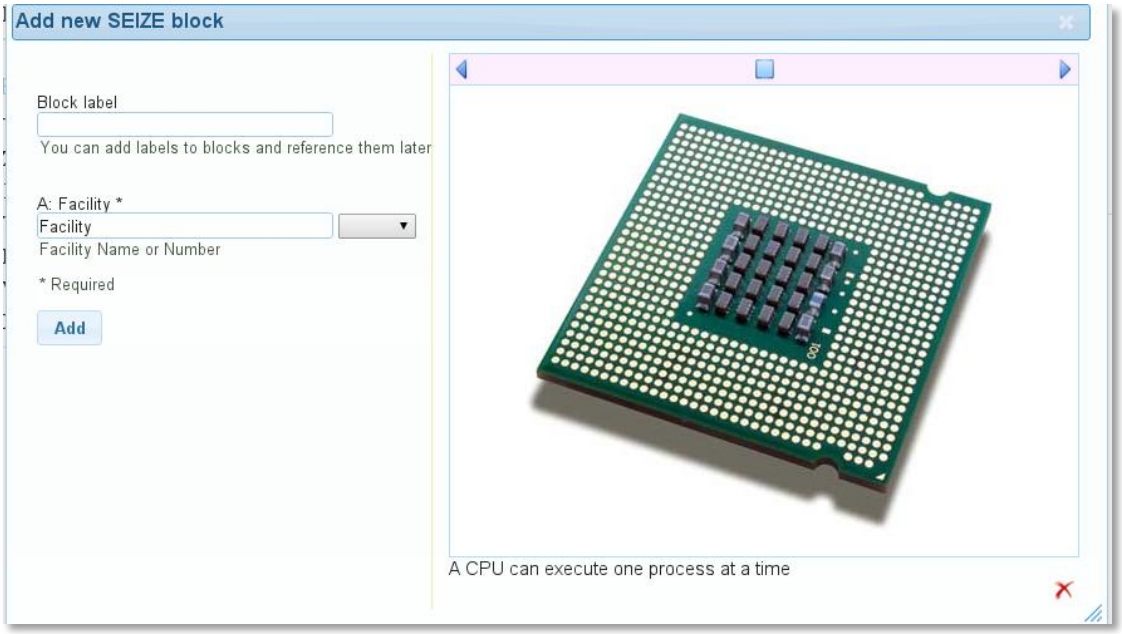


Imagen 44: Imagen de una CPU como ejemplo de Facilidad

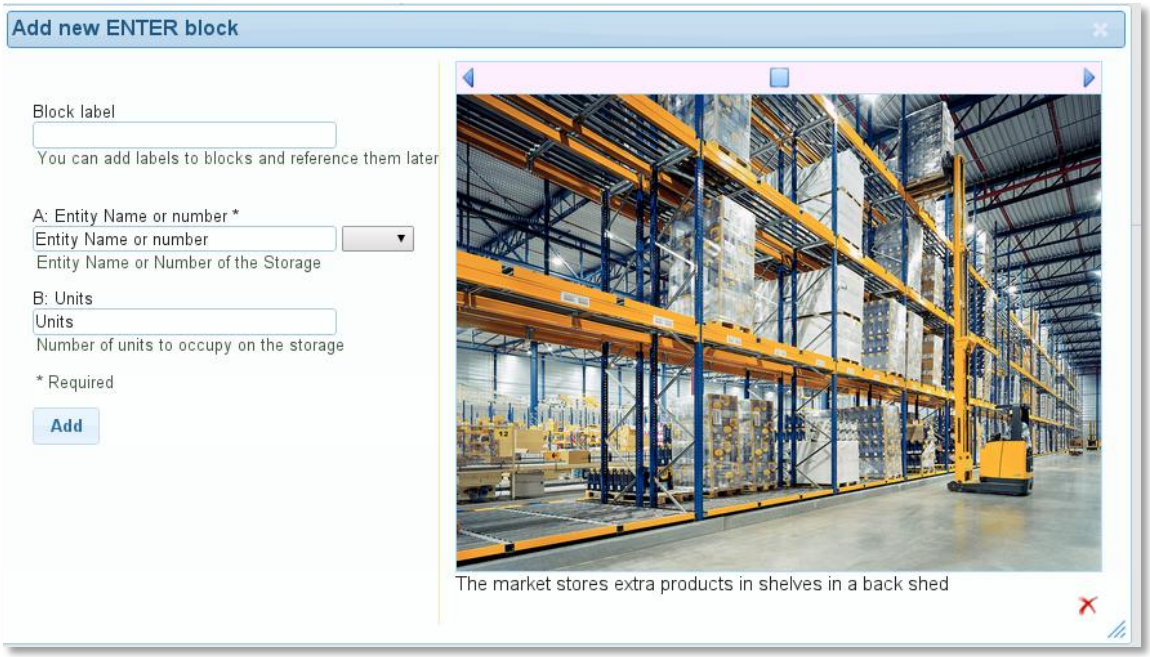


Imagen 45: Fotografía de un depósito como ejemplo de Almacén

Ejecución de modelos y visualización de resultados

Una vez que los alumnos han construido sus modelos, deberán ejecutarlos en el simulador para ver cómo funcionan y qué resultados generan. Para ello, la pantalla principal presenta una barra superior con un botón que lanza la ejecución de la simulación (botón "Play"). Dentro de la misma barra también se incluyen algunos accesos rápidos para generar modelos pre-armados con un solo clic. Esto permite al docente crear algunos modelos particulares que destacan algún aspecto puntual, o crear ejercicios para que los alumnos analicen, mejoren o corrijan (Imagen 46). Los modelos generados desde aquí funcionarán de manera idéntica a aquellos generados manualmente por los alumnos. Si bien no se ha incluido hasta aquí una interfaz de carga de modelos, existe un módulo dentro de GPSS Interactivo dedicado a la inclusión de modelos de pruebas y modelos listos para usar (/tests/test.js), y su creación sólo requiere unas pocas líneas en Javascript que especifique bloques, comandos y parámetros.

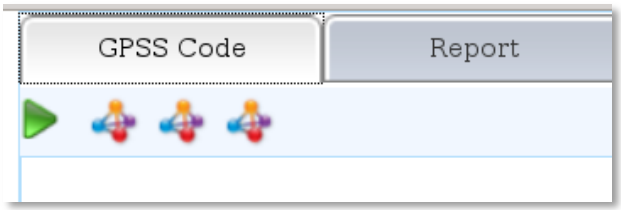


Imagen 46: Botones para ejecutar el modelo y para cargar modelos preexistentes

La pantalla de resultados se divide en dos secciones principales (Imagen 47): por un lado, a la izquierda de la pantalla se presentan todos los tiempos de reloj por los que pasó la simulación. Al hacer clic sobre cada uno de estos "tiempos de simulación", el sistema cargará automáticamente la *captura instantánea* de la simulación en ese momento; además, se resalta el tiempo actual de simulación que se está visualizando del resto de los tiempos de simulación. Por el otro lado, a la derecha de la pantalla, se visualiza el estado de la simulación en un instante determinado. Esto incluye el estado de las cadenas, de las entidades permanentes y de las entidades temporales que existen en este momento.

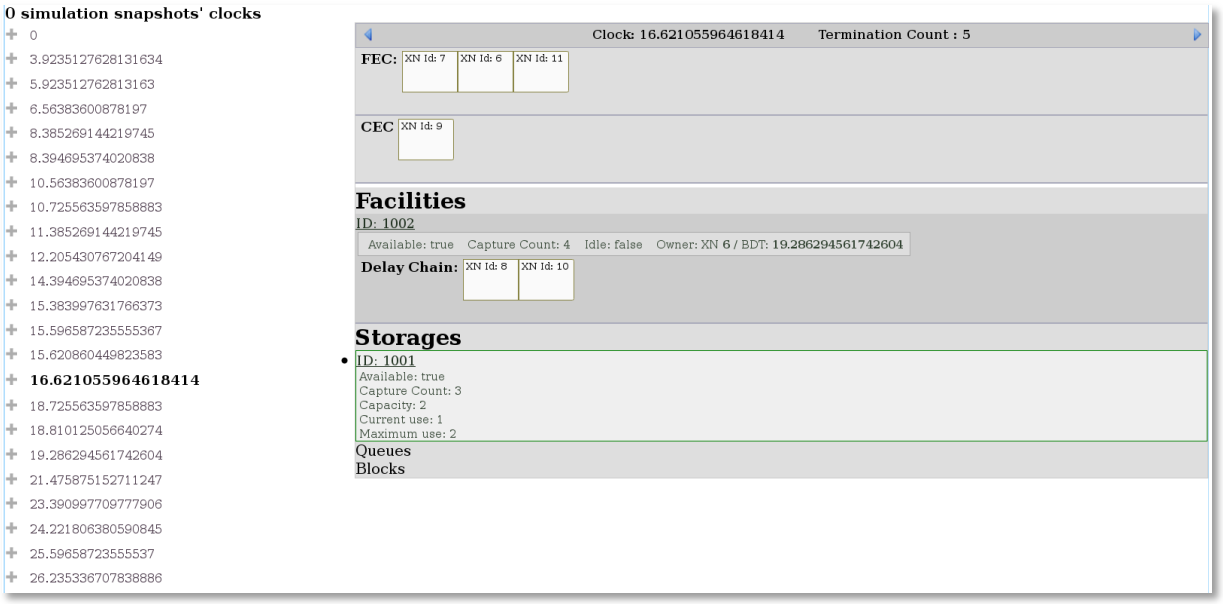


Imagen 47: Vista general de la pantalla de resultados de simulación

En la parte superior de esta sección se localiza la Barra de progreso (Imagen 48). Esta barra permite al usuario avanzar o retroceder secuencialmente en la navegación del historial de simulación. Si bien esto también puede hacerse desde el listado de la izquierda con todos los tiempos de simulación, es más simple realizar así una navegación secuencial rápida para el usuario, mientras se destaca la idea de orden o secuencialidad (tiempo siguiente, tiempo anterior). Asimismo, al centro de la barra se muestra el tiempo actual de simulación que se está visualizando y, más importante, el valor del contador de terminación de la simulación (*Termination Count*), que le permite al usuario tener una primera idea del grado de avance de la simulación hasta este punto.

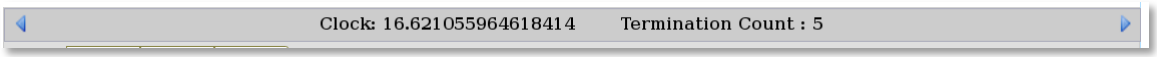


Imagen 48: Barra de progreso

Debajo de la barra de progreso se presentan al usuario las dos principales cadenas del sistema (Imagen 49): la Future Events Chain (FEC) y la Current Events Chain (CEC). Esta vista destacada permite apreciar a simple vista la importancia de estas cadenas en la ejecución de la simulación, y además acentúa algunos conceptos fundamentales de las mismas como su composición por transacciones y la organización de las mismas según determinados criterios propios de cada cadena (orden de llegada, prioridad, tiempo de

partida del bloque).

Cada transacción se identifica del resto por su identificador interno (ID), asignado secuencialmente de manera automática por el motor de simulación al momento de crearla. El orden de las cadenas es siempre el mismo: la primera transacción es la que se muestra más a la izquierda, y la última es aquella que está más a la derecha. Dado que la captura de la simulación se realiza antes de comenzar a ejecutar las transacciones listas para cada tiempo de reloj de simulación, la cadena CEC siempre tendrá al menos una transacción lista. Como ya se explicó, la generación de capturas instantáneas de la simulación no se realizó al finalizar la ejecución de las transacciones en cada tiempo de simulación, pues esto provocaría que la CEC estuviese siempre vacía, y sería por lo tanto muy difícil reconocer qué transacciones fueron ejecutadas durante este instante de simulación y cuáles ya se encontraban en las distintas cadenas del sistema desde antes.

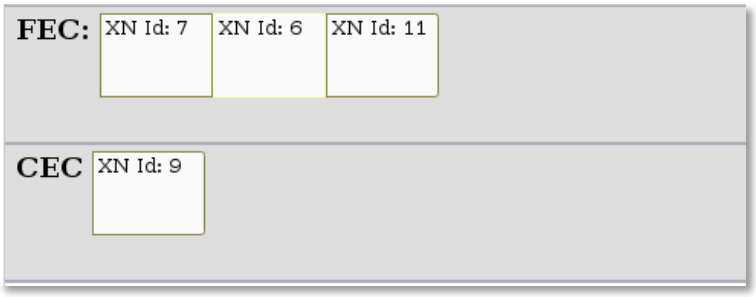


Imagen 49: Vista de las cadenas FEC y CEC en un instante de simulación

Al seleccionar una transacción, el sistema desplegará toda la información que contiene sobre la misma. Esto permite al usuario obtener una vista más detallada de los parámetros internos y los parámetros asignados por él mismo (Imagen 50). Además, un análisis de algunos parámetros de la transacción permite al usuario comprender por qué la misma se encuentra allí, por cuánto tiempo más, y qué sucederá cuando sea ejecutada por el simulador. Estos parámetros son la prioridad (priority), el tiempo de partida del bloque (BDT), el bloque actual (Current Block) y el siguiente bloque secuencial (NSB). Otros parámetros, como por ejemplo el indicador de retraso (Delay Indicator) o el contador de bloques (Block Count) permiten también estimar cómo ha sido la ejecución de la transacción hasta aquí. Durante las pruebas, como se verá en el siguiente capítulo, estas características fueron muy útiles para reconstruir la ejecución de la simulación

hasta cierto punto, y así recrear con los alumnos el funcionamiento del simulador.

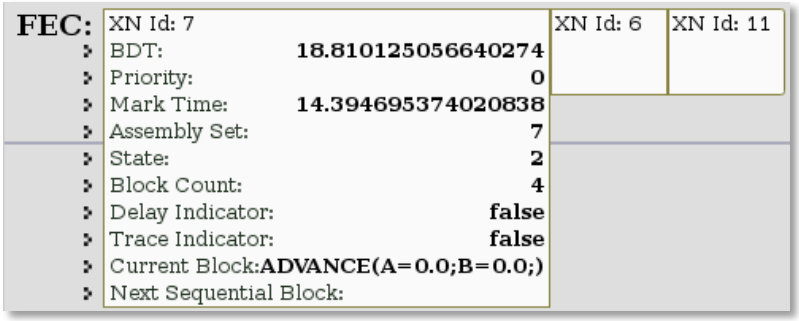


Imagen 50: Vista detallada de una facilidad en la cadena FEC en un instante de simulación

En la zona inferior de la pantalla de resultados de simulación se muestran las entidades permanentes creadas durante la simulación, y el estado de las mismas en el instante de reloj actual. Las primeras entidades permanentes que se muestran son las Facilidades (Imagen 51), una debajo de la otra. Para cada facilidad, se muestra su identificador interno (ID), y el estado de sus variables de estado internas: disponible (*available*), contador de entradas (*capture count*), libre (*idle*), y el identificador de la transacción "dueña" (*owner*) en ese momento. Además, se incluye aquí el tiempo de partida de dicha transacción de su bloque actual (BDT), lo cual permite al alumno estimar —suponiendo que no habrá preempciones ni interrupciones— durante cuánto tiempo esa transacción permanecerá como *owner* de la facilidad, y cuál debería tomarla a continuación. Para completar la información de la facilidad, se muestran una a una las cadenas de la misma que poseen al menos una transacción: Delay Chain, Retry Chain, Interrupt Chain y Pending Chain. La visualización e interacción sobre las cuatro cadenas es igual en todos los casos, y también se replica la visualización e interacción las cadenas FEC y CEC, lo cual mantiene uniformidad en la forma de interpretar cualquier cadena, y permite al usuario acceder siempre a toda la información relativa a cada una de las transacciones en las cadenas.

De manera similar a las facilidades, los almacenes se muestran (Imagen 52) destacando sus ID internos, y exponiendo los valores de sus atributos internos: disponible (*available*), contador de entradas (*capture count*), capacidad máxima (*capacity*), uso actual (*current use*), uso máximo (*maximum use*). Para finalizar la vista del almacén, se muestran sus cadenas Delay Chain y Retry Chain (si las mismas poseen

transacciones para mostrar).

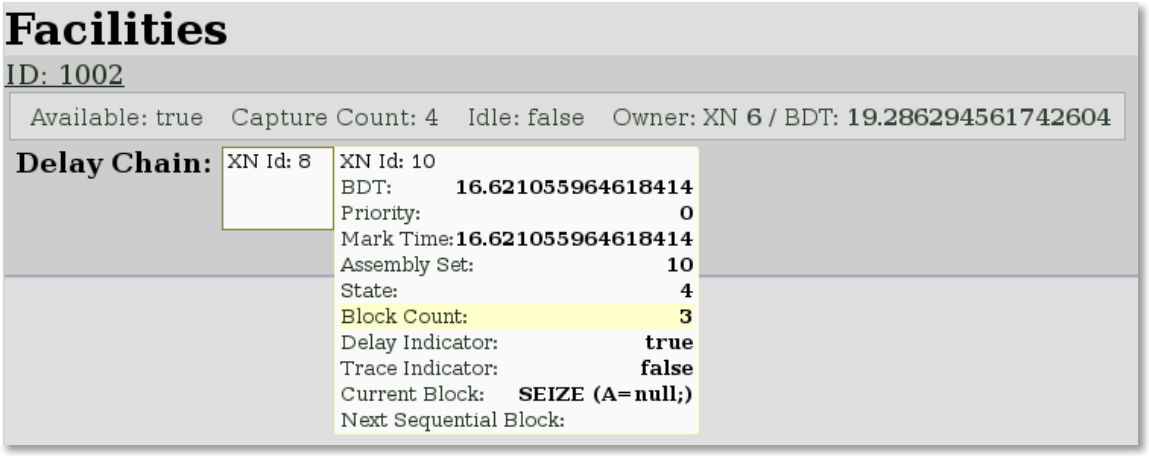


Imagen 51: Vista de una facilidad en un instante de simulación

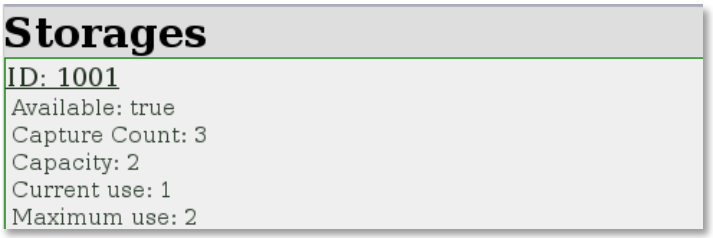


Imagen 52: Vista de un almacén en un instante de simulación

Diseño del curso

En el capítulo anterior se analizaron en detalle las principales características de GPSS Interactivo, tanto desde el punto de vista de su diseño orientado a la enseñanza como desde la arquitectura, módulos y tecnologías aplicadas durante el desarrollo. Si bien esta herramienta es una parte fundamental de la propuesta de enseñanza que se presenta en este trabajo, está claro que por sí sola no es suficiente para garantizar un aprendizaje correcto. Para su mejor aprovechamiento, fue necesario realizar una planificación del curso de Modelos y Simulación ajustada a las necesidades e intereses de los alumnos a medida que avanzaban las clases. En esta parte de este capítulo se explica precisamente en qué consiste este rediseño del curso con la herramienta GPSS Interactivo.

El curso contó con la participación de alumnos de las carreras de sistemas, y la propuesta incluyó ejemplos sobre cómo se llevaron a cabo durante las clases las

explicaciones, descripciones y ejercitaciones. En la última parte del capítulo se enumeran los principales resultados alcanzados tanto desde el punto de vista de la experiencia como de los resultados obtenidos en las evaluaciones.

Organización temática

El diseño de un curso de Modelado y Simulación es, al igual que en la mayoría de las asignaturas, un factor decisivo que determina la organización temática y duración del curso, y que a su vez influye en el éxito o fracaso del mismo a partir del conocimiento adquirido por los alumnos. Los cursos en esta área se destacan por poseer una fuerte impronta tecnológica, caracterizada por la combinación de conceptos teóricos sobre modelado, sistemas y estadísticas, junto con la introducción de al menos una herramienta de software que permite aplicar dichos conceptos en un contexto controlado. Estas herramientas son utilizadas para desarrollar los modelos, ejecutarlos, experimentar y analizar resultados junto a los alumnos del curso. En este contexto, cabe señalar que los modelos sobre los que se trabaja en el curso son por lo general versiones simplificadas de modelos más complejos, que sirven o bien para estudiar determinados aspectos puntuales de los sistemas que representan, o bien para aplicar y confirmar ciertos conceptos vistos dentro del marco teórico.

Las herramientas informáticas que suelen utilizarse para simular modelos y ejecutarlos son por lo general aplicaciones profesionales avanzadas y muy completas. Cuentan con gran cantidad de funciones disponibles para el desarrollador, que permiten crear modelos, ejecutarlos paso a paso (*debugging*), implementar rutinas propias y controlar la repetición de la ejecución de los modelos (por ejemplo, mediante el lenguaje PLUS en el caso de GPSS), diseñar experimentos de distintos tipos, realizar análisis de varianza (ANOVA), entre otras funciones. Como sucede con cualquier aplicación de estas características, las herramientas de simulación son realmente aprovechadas con todo su potencial cuando se las conoce en profundidad, ya que las soluciones que se implementan se pueden pensar desde un principio para sacar provecho de las funciones disponibles en el entorno.

Como ya se estudió en el capítulo 3, GPSS es una herramienta de simulación completa, ampliamente utilizada en el ámbito profesional y académico. GPSS hace referencia tanto

al lenguaje de programación como al simulador en sí. El simulador está compuesto por un conjunto de entidades predefinidas y un motor de simulación que determina la manera en que estas entidades interactúan entre sí y las rutinas que cada una ejecutará. El lenguaje de programación, por otro lado, permite al desarrollador realizar modelos que utilicen dichas entidades y aprovechen las facilidades que el motor de simulación ofrece. Si bien GPSS es muy utilizado para la enseñanza de modelado y simulación por eventos discretos y de sistemas gobernados por colas, no deja de ser una aplicación profesional compleja, y por lo tanto queda sujeto a las mismas reglas que el resto de las aplicaciones de este tipo. El diseño de un curso debe por lo tanto considerar las características de la o las herramientas con las que se trabajará, a fin de combinar los aspectos teóricos propios del área con las cuestiones más prácticas, aplicables directamente sobre las herramientas de trabajo, pero también debe evitar una sobrecarga de información desorganizada que afecte de manera negativa el proceso de aprendizaje de los alumnos del curso.

En cuanto a la organización temática tradicional, los cursos y textos de simulación en general, y de GPSS en particular [Karian98], se organizan a partir de la enseñanza progresiva del lenguaje de programación en cuestión, basándose en una constante evolución en cuanto a la complejidad de las entidades y su interacción con otros elementos del lenguaje previamente vistos. Una vez que se han dictado los principales conceptos generales del lenguaje, se busca que el alumno comprenda cómo interactúan todos estos elementos por detrás, o sea cómo funciona realmente el simulador por dentro y cómo están representados internamente los elementos vistos durante las clases previas. La introducción de estos temas brinda a los alumnos una visión más integral del simulador GPSS, y otorga un mayor significado a los conceptos previamente estudiados. Además, al conocer internamente la herramienta con la que trabajan, se adquiere un mayor criterio a la hora de elegir entre distintas soluciones frente a un mismo problema, y de seleccionar las herramientas más adecuadas para el trabajo en el ámbito profesional. Para ello, los temas del curso avanzan hacia el funcionamiento interno del motor GPSS, tanto desde el punto de vista de compilación e inicialización de una simulación, como desde la ejecución de las sentencias que conforman la simulación en sí.

Luego de haber estudiado y comprendido el funcionamiento del motor de simulación de GPSS por dentro, los alumnos por lo general logran atar todos los cabos sueltos de clases anteriores, y comprender mejor cuáles son las motivaciones que generaron ciertas decisiones de diseño e implementación de GPSS. Este mecanismo es comparable a la enseñanza de programación desde los primeros años de las carreras de sistemas, donde se parte desde los conceptos más básicos como variables, constantes y sentencias del lenguaje, acompañado por ideas más abstractas como la sintaxis y la semántica del lenguaje, para luego avanzar hacia conceptos como interpretación de código fuente, compilación a código de máquina y ejecución de dicho código, y finalmente comprender cómo los programas son ejecutados en el contexto del Sistema Operativo, incluyendo otros procesos, prioridades y planificación (*scheduling*) de la ejecución.

La organización temática típica arriba descripta supone que los alumnos finalizarán los cursos de Modelos y Simulación con una base amplia de conocimiento del área y sobre esta herramienta de trabajo en particular. Sin embargo, esta estructura también provoca que durante la mayor parte del curso los alumnos tiendan a adoptar un rol principalmente pasivo, escuchando y creyendo de buena fe que los conceptos que el docente imparte son ciertos, y que las entidades y rutinas del simulador funcionan "como por arte de magia". Los alumnos ejercen el oficio de alumnos en relación tanto con el conformismo como con la competencia [Sirota, 2000]. Durante la interacción entre los alumnos y el docente, la influencia educativa del docente se ejerce a través de un proceso que abarca la actividad constructiva del alumno por un lado, y la actividad del docente por el otro, con su capacidad para orientar y guiar la actividad del alumno hacia la realización de los aprendizajes. En este sentido, la enseñanza puede ser descripta como un proceso continuo de negociación de significados, de establecimiento de contextos mentales compartidos [DeVargas06]. En esta negociación, los alumnos reciben pasivamente los conceptos que los docentes imparten, sin analizarlos o discutirlos demasiado. No obstante, es de esperar que una actitud más activa frente a las clases fomentará la interacción y el debate sobre ciertos temas, permitiendo por ejemplo profundizar algunos puntos de mayor interés, o incluso discutir alternativas a los diseños o implementación propuestos, lo cual ayuda a enriquecer mucho más el dictado del curso tanto para los alumnos como para los docentes, y finalmente hará que los temas aprendidos sean registrados de manera más sólida en las mentes de los alumnos.

Sin embargo, bajo un esquema de curso caracterizado principalmente por un rol pasivo y un aprendizaje casi exclusivamente por recepción por parte de los alumnos, sólo cuando han acumulado buena parte de los conceptos y han estudiado el simulador por dentro, éstos logran construir una base sólida de conocimiento que le da sentido a lo aprendido previamente, y se pierde de este modo la posibilidad de participar y oportunidad de enriquecerse de los debates con los docentes y con sus compañeros.

Además del rol (esperable) de los alumnos y del docente dentro de las clases, la organización de los temas del curso puede también influenciar tanto favoreciendo como obstaculizando la participación activa de los alumnos. A lo largo de un curso de Modelos y Simulación típico suele seguirse una organización metodológica desde lo particular a lo general (*bottom-up*), en la cual se imparten los distintos temas en un aprendizaje por partes, aportando paulatinamente nuevos conceptos cada vez más complejos, hasta que la suma de todos los conceptos permite cubrir todos los temas del curso. Por ejemplo, en el caso de GPSS, se introducen nuevas entidades de manera paulatina, y se construyen modelos cada vez más complejos hasta que llega un punto en el cual los alumnos deben comprender cómo ejecuta el motor de simulación estas entidades para así poder desarrollar modelos más avanzados y complejos.

Por otro lado, una característica que se presenta bajo este esquema es que requiere una importante inversión de tiempo para explicar y lograr la asimilación de todos los conceptos nuevos, especialmente durante la primera parte del curso. A medida que se introducen contenidos cada vez más complejos y abstractos (conceptos teóricos, entidades avanzadas, nuevos bloques, comandos, etcétera), la relación entre los nuevos temas y los vistos previamente no es aún significativa para los alumnos, y no lo será hasta que no logren adquirir una visión integral del simulador y su forma de trabajo. Con esta ausencia de una visión integral, se hace más difícil para los alumnos asignarle significado a los nuevos conceptos y conectarlos con otros ya vistos por sí mismos. Este trabajo recae necesariamente sobre el docente del curso, quien debe forzar el descubrimiento de las relaciones conceptuales, y se ve acentuado en particular durante las primeras clases del curso, en las cuales el nivel de conocimiento general de los alumnos es aún demasiado escaso para generar dudas o inquietudes a medida que adquieren el nuevo conocimiento.

El tema de la generación, ejecución y finalización de transacciones resulta útil para ejemplificar esta situación. Este tema es, por lo general, uno de los primeros que se ven durante la cursada. Se introducen los tres bloques más básicos del lenguaje (básicos en relación la cotidianidad de su uso, y no por su facilidad): GENERATE, ADVANCE y TERMINATE. Se les explica a los alumnos que existe un comando, START, que sirve para iniciar la simulación. Desde luego, los alumnos apenas conocen la diferencia entre un bloque y un comando, con lo cual esto ya comienza a generar ciertas dudas. El concepto de tiempo entre generación y tiempo de avance tampoco queda claro aún, pues a esta altura del curso no están familiarizados con la idea de un reloj de simulación y el tiempo de simulación. Desde luego, la creación y ejecución de transacciones es otra de las áreas más oscuras, pues no existen aún en sus mentes los conceptos de cadenas, planificación de transacciones, etcétera. A continuación se listan algunas de las dudas más comunes que surgen durante las primeras clases:

- ✓ ¿Qué significa el número que acompaña al comando START?
- ✓ ¿En qué afecta al modelo si el comando START está más arriba o más abajo?
- ✓ ¿Qué pasa si se generan dos transacciones juntas?
- ✓ ¿Las transacciones concurrentes se ejecutan como procesos independientes o como hilos de un mismo proceso?
- ✓ ¿Quién "despierta" a las transacciones que fueron "dormidas" por el comando ADVANCE?
- ✓ ¿Qué pasa si se terminan todas las transacciones?

Desde luego, todas estas dudas son naturales a esta altura, pero la mayoría o bien tienen una respuesta muy obvia, o bien carecen de sentido debido a que los alumnos están tratando de asociar lo que acaban de aprender con conceptos adquiridos previamente en otros cursos. El principal problema aquí no es que surjan estas dudas, pues el docente siempre puede intentar explicarlas, al menos en una versión muy simplificada. El problema es que los alumnos aún no están preparados para comprender por completo muchas de estas cuestiones, y el docente deberá formular una respuesta ahora, para luego —en las clases siguientes— cambiar el discurso a fin de responder a estas mismas preguntas basándose en las nuevas herramientas y conceptos con los que

contarán los alumnos.

Recién a mediados de la cursada los alumnos tendrán un panorama lo suficientemente amplio como para comenzar a participar de manera un poco más activa en la clase, planteando alternativas y proponiendo ideas propias. Esto demuestra que los alumnos están empezando a comprender y relacionar los distintos temas, ya que comienzan a entablar correctamente comparaciones y paralelismos con conceptos adquiridos previamente en otros cursos así como también con conceptos vistos a lo largo de este curso, plantear dudas sobre cuestiones más profundas, e incluso cuestionar ideas o implementaciones propuestas. Se observa también un incremento en la velocidad de aprendizaje, así como también una mayor facilidad en la adquisición de los nuevos temas, potenciado por la visión más abarcativa y por la mayor participación de los alumnos en la clase. Sin embargo, existe una evidente pérdida de oportunidades de interacción y diálogo durante las clases iniciales, en las cuales el proceso de aprendizaje fue más lento y menos participativo.

Adicionalmente, se observa también en las instancias finales del curso que, a lo largo de las distintas clases, se ha dedicado mucho —quizás demasiado— tiempo a intentar comprender cómo usar una herramienta tecnológica particular, con el objetivo de que los alumnos puedan conocerla a fondo y aprovecharla como se espera. Dicho de otro modo, se le otorga demasiado protagonismo al software de simulación —GPSS en este caso—, y se corre el riesgo de convertir un curso de Modelos y Simulación en un taller avanzado en un lenguaje de simulación como GPSS que, al fin y al cabo, es sólo un lenguaje más. En consecuencia, al centralizar la atención sobre la herramienta de trabajo, se pierde por falta de tiempo la oportunidad de profundizar en cuestiones más avanzadas como metodologías de modelado, ejemplos de sistemas y modelos más complejos, y diseño de experimentos para estudiar mejor los modelos desarrollados.

Otro inconveniente que surge a partir de la ausencia de una visión integral inicial es que afecta negativamente la participación de los alumnos en clase. Al desconocer internamente los diseños y mecanismos de la herramienta que los alumnos están aprendiendo a utilizar, se les hace más difícil suponer o proponer soluciones a los distintos problemas que se presentan durante las clases. La falta de un rol activo no incentiva a los alumnos a realizar preguntas, no fomenta el debate con sus pares y

docentes, y principalmente no promueve la generación de conflictos cognitivos que servirán luego para afianzar conceptos teóricos y prácticos. Bajo estas condiciones, se dificulta la realización de estrategias docentes centradas en el aprendizaje experiencial y situado, como el Aprendizaje Centrado en la Solución de Problemas Auténticos y el Análisis de Casos [ElainedeVargas06], que se enfocan en la construcción del conocimiento en contextos reales, en el desarrollo de las capacidades críticas, reflexivas y en el pensamiento de alto nivel [Díaz03]. Esta barrera puede ser superada por lo general una vez que el curso ha avanzado lo suficiente para que los alumnos puedan integrar conceptos por sí solos, pero para entonces ya se han invertido muchas horas y se han perdido muchas oportunidades de interacción alumno-alumno y alumno-docente.

De lo particular a lo general

Con el objetivo principal de fijar los conceptos desde el primer momento en que se brindan y lograr así un mayor grado de significación en el aprendizaje, pero también de aprovechar mejor el tiempo dedicado a enseñar el uso del simulador GPSS con el fin de ampliar la diversidad y profundidad en los temas incluidos en las clases, se ha planteado un rediseño del curso de Modelos y Simulación. Este rediseño fue basado en una reorganización de los temas que se brindan a lo largo curso, e incluyó a la herramienta GPSS Interactivo desarrollada *ad hoc* como soporte de aprendizaje, especialmente durante los primeros encuentros de los alumnos con esta herramienta.

CLASE	ORGANIZACIÓN TEMÁTICA TRADICIONAL	ORGANIZACIÓN TEMÁTICA PROPUESTA
1	Sistemas, subsistemas y componentes; modelos, tipos de modelos, modelos de simulación; tipos de simulaciones, lenguajes de simulación vs tradicionales. Metodología para plantear una simulación: modelado, identificación de entidades, flujo de control, elección de un lenguaje, validación... Entidades permanentes vs entidades temporales.	Sistemas, subsistemas y componentes; modelos, tipos de modelos, modelos de simulación; tipos de simulaciones, lenguajes de simulación vs tradicionales. Metodología para plantear una simulación, muy simplificada. Entidades permanentes vs entidades temporales. Planteo de un modelo de cola simple, y ejecución en una tabla del pizarrón.

CLASE	ORGANIZACIÓN TEMÁTICA TRADICIONAL	ORGANIZACIÓN TEMÁTICA PROPUESTA
2	Planteo de un modelo de cola simple, y ejecución en una tabla del pizarrón. Visión general de GPSS: facilidades y sus bloques, storages y sus bloques, funciones y tipos de funciones, logic switches, colas y sus bloques, retardos, manejo de atributos, SNA, compilación del modelo, ejecución y análisis del reporte.	Presentación de facilidades y transacciones. Asociación con elementos del ejemplo de la clase anterior. Entidad STORAGE. El concepto de bloques y comandos. Bloques GENERATE, TERMINATE, ADVANCE, ASSIGN, BUFFER, SEIZE, RELEASE, ENTER, LEAVE. Bloques y entidades enseñados a partir de la herramienta interactiva. Análisis detallado de su ejecución de un modelo en GPSS. Entidades del sistema: FEC; CEC; Clock; Termination Count. Planificación de transacciones completa. Múltiples generadores; simulación controlada por reloj.
3	Creación de un primer modelo simple, ejecución, comando START, parámetros del generate, relación con el termination count.	Entidad QUEUE, bloques QUEUE y DEPART. Análisis de una corrida en detalle. Bloques SAVEVALUE, GATE, TEST, TRANSFER (pocos modos). El problema de la retry chain, USERCHAIN, bloques LINK/UNLINK. Conceptos básicos de debug.
4	Bloques TRANSFER (todos los modos), TEST, GATE.	FUNCTION, tipos y definición. PREEMPT (pending e interrupt chain). Assembly Set, SPLIT, MATCH, GATHER, ASSEMBLE. Ejemplo integrador.
5	El problema de la retry chain. USERCHAINS, LINK; UNLINK. GOUPS, JOINS, REMOVE,...	GROUPS, JOIN, REMOVE. Debug en GPSS.
6	Análisis detallado de su ejecución de un modelo en GPSS. El planificador de transacciones.	
7	PREEMPT (Pending e interrupt chain). MATCH, GATHER, ASSEMBLE, SPLIT.	

CLASE	ORGANIZACIÓN TEMÁTICA TRADICIONAL	ORGANIZACIÓN TEMÁTICA PROPUESTA
8	Ventanas de GPSS. Debug. Ejemplo integrador.	

Tabla 1: Organización temática tradicional versus organización temática propuesta

Dentro de la nueva organización temática se incluye al inicio del curso un módulo introductorio que contiene básicamente un repaso de ciertos conceptos teóricos que los alumnos por lo general ya conocen y que sirven para darle sentido al curso. Este módulo posee los mismos contenidos del módulo original, y su objetivo es que los alumnos repasen ciertos conocimientos previos mínimos que servirán como sustento para el resto de los contenidos. Dentro de este módulo se incluyen varios aspectos de la Teoría General de Sistemas, los distintos tipos de sistemas, conceptos de modelado y tipos de modelos, usos de la simulación como herramienta, tipos de simulaciones, y algunos ejemplos de simulaciones "reales". Si bien los alumnos por lo general están familiarizados con estos temas, se busca una presentación de contenidos más general y menos atada a aspectos informáticos. Así, por ejemplo, se toman casos reales de ámbitos muy diversos como arquitectura —considerando planos en 2D y maquetas como diferentes modelos abstractos de un mismo *sistema casa*— o ingeniería aeroespacial —mediante modelos a escala de aeronaves—. El objetivo aquí es ampliar el concepto de sistema-modelo más allá del entorno del software, mover a los alumnos del entorno informático al que están acostumbrados, para que comiencen a ver la aplicabilidad de la simulación como herramienta en otros ámbitos. Este criterio se mantiene también a lo largo de la cursada, combinando modelos y prototipos basados en sistemas informáticos con otros tomados de otras áreas diversas.

La última parte del módulo introductorio se centró en un caso basado en un sistema concreto, en el cual los alumnos debían considerar una línea de envasado con un sistema de clasificación automática de frutas. Esta clasificación se realizaba mediante un ojo robótico (Anexo 1), y las frutas se movían dentro del sistema utilizando una cinta transportadora. En este sistema, el objetivo principal era que los alumnos comenzasen a familiarizarse con lo que luego reconocerán como un sistema con una única cola y un

único servidor, y que empezasen a acostumbrarse a explicar el comportamiento de los sistemas a partir de cierta terminología específica como tiempo entre llegadas, tiempo de servicio, tiempo de tránsito, colas de espera, etcétera. Durante este primer ejercicio grupal, se buscó también fomentar la participación de los alumnos en la discusión, en especial cuando se trataba de identificar las entidades del sistema (el robot, las frutas, la cinta) y sus características principales. Como verán más adelante los alumnos, existe una correlación directa entre dichas características, algunos conceptos de simulación, y varios de los elementos de GPSS (tabla 2).

Para fomentar la participación en clase, el docente solicitaba una y otra vez que los alumnos describan el sistema, profundizando cada vez más el nivel de detalle, mientras tomaba nota en una pizarra de todo aquello que consideraba relevante. Nuevamente, también se trazaron aquí paralelismos con cuestiones de índole más informática, comparando por ejemplo el robot con un procesador de computadoras, y las frutas con procesos del sistema operativo que utilizan dicho procesador. Aquí, el docente sólo planteaba la existencia de un paralelismo, y los alumnos debían identificar en qué consistía, cuáles eran los elementos comunes y porqué. Lo importante de estos paralelismos era forzar el reconocimiento de estos aspectos comunes, para más adelante generar una visión abstracta de los distintos tipos de entidades de los sistemas.

En esta instancia del curso, la diferencia más destacable fue el permanente incentivo al diálogo y a la participación desde el docente hacia los alumnos, buscando que éstos tomen una posición más activa en el proceso de aprendizaje. Era importante aquí que los alumnos comenzasen a reconocer este dinamismo como parte del curso, hasta que lograsen adoptarlo como natural. El rol del docente fue clave aquí, ya que debía forzar, en varias situaciones, la participación de los alumnos, a fin de que pierdan la timidez inicial propia de gran parte de los estudiantes, de *romper el hielo*, pero también para otorgarle un mayor sentido al análisis que estaban realizando.

SISTEMA DE CLASIFICACIÓN	SIMULACIÓN	GPSS
El robot sólo puede analizar una fruta por vez	Servidor simple	Entidad Facility
El estado del robot puede ser libre u ocupado	Servidor libre u ocupado	Bloques SEIZE, RELEASE
Las frutas llegan al sistema con cierta periodicidad	Tiempo entre arribos, generadores de números pseudo-aleatorios	Bloque GENERATE, random number generator
Las frutas salen del sistema una vez que han sido analizadas por el robot	Finalización de entidades, tiempo de terminación de la simulación	Bloque TERMINATE, termination_count, comando START
La cinta controla la interacción entre las frutas y el robot, rige el tiempo que les toma moverse desde que ingresan al sistema y la velocidad que permite al robot observarlas y tomar una decisión	Tiempo de servicio, encolamiento de eventos	Bloque ADVANCE, cadenas FEC y CEC

Tabla 2: Relación entre las características del sistema del ojo robótico, los conceptos de simulación, y los elementos de GPSS

En los siguientes encuentros comenzó la reestructuración temática más importante. En el siguiente módulo, inmediato a la introducción teórica, se brindaron los conceptos relativos al simulador de GPSS por dentro. Dicho de otra manera, se les explicó a los alumnos en qué consistía exactamente el motor de simulación, cómo y por qué funciona como funciona. Para introducir estos conceptos, el docente debía acotarlos a un conjunto limitado de entidades —transacciones, facilidades, almacenes y colas—, e impartir una visión algo intuitiva y simplificada de las mismas. El objetivo aquí no era comprender la totalidad de las entidades, sino hacer especial hincapié en el mecanismo de planificación de transacciones que realiza el simulador, en los sistemas de gestión de colas y en la función que tienen las principales colas o cadenas de GPSS. También se explicó cómo se interpretan las distintas porciones de código GPSS —los comandos por un lado y los bloques por el otro—, qué estructuras internas utiliza el motor de simulación para alojar

las distintas porciones de código y, finalmente, cómo se inicia, ejecuta y termina una simulación. Este cambio metodológico, que va desde lo general a lo particular (*top-down*), brindó un panorama mucho más amplio desde el comienzo, que luego fue desglosado paulatinamente con el fin de incorporar nuevos conceptos y conocer en detalle cada una de las partes de la visión global. Si bien, en el punto inicial, los conceptos eran bastante avanzados y por lo general resultaron algo complejos de comprender, esta estructura de contenidos invertida permitió cimentar las bases de GPSS desde el comienzo, y sobre ellas luego se pudieron construir los nuevos conocimientos.

Durante las siguientes clases, se continuó con la incorporación de nuevos elementos de GPSS y con la profundización de aquellos ya vistos, siempre buscando analizar cómo se relacionaban los nuevos conceptos con los anteriores, cómo interactuaban entre sí durante la simulación, y en especial cómo hacía el simulador GPSS para trabajar con los nuevos elementos. Es importante destacar que los nuevos conceptos no se impartían —siempre que fuera posible— de manera directa, sino que por lo general se partía desde un problema particular, que los alumnos debían analizar y proponer luego soluciones utilizando todos los elementos con los contaban hasta ese momento; estos ejercicios ponían al descubierto los conflictos cognitivos, necesarios para posteriormente derivar los nuevos conceptos y darles un mayor significado a los mismos. Esta forma de llevar a cabo las clases, si bien requirió más tiempo y esfuerzo por parte del docente y de los alumnos comparado con un sistema más directo —en el cual simplemente el docente presenta los temas nuevos — persigue tres propósitos específicos:

1. que los alumnos participen activamente y repasen los conceptos ya vistos al buscar la solución a un problema basados en dichos conceptos;
2. que se revele la necesidad de nuevos elementos que permitan o simplifiquen la solución del problema planteado, a partir de conflictos cognitivos generados en los alumnos;
3. que sean los alumnos quienes descubran y describan (al menos intuitivamente) estos nuevos elementos, a partir de las soluciones que el simulador propone para los elementos que ellos ya conocen.

El rol del docente se enfocó en destacar los puntos positivos de las propuestas y

comentarios de los alumnos, y en orientarlos durante su participación para mantener la dirección apropiada. De este modo, si los alumnos eran capaces de detectar la necesidad de nuevos elementos, nuevas características y/o nuevas herramientas, y posteriormente podían aproximar una solución basándose en el conocimiento de otras soluciones a otros problemas, el aprendizaje sería más significativo. El nuevo conocimiento no había sido impuesto, sino que fue inferido lógicamente por ellos mismos, construido a partir del conocimiento previo. A continuación, se describen dos ejemplos que explican cómo se puso esto en práctica.

Ejemplo 1: el comando STORAGE

Un ejemplo de este conocimiento inferido surgió al momento de aprender la entidad de GPSS denominada STORAGE. Los alumnos ya conocían la entidad FACILITY, y en este punto del curso se les había comentado la existencia de una entidad que posee un "tamaño" o cantidad de unidades utilizables.

Como es de esperarse, este valor de tamaño no es único y fijo para dicha entidad, sino que debe definirse cada vez que se instancia una entidad de este tipo en cada modelo diferente. Por lo tanto, dado que GPSS sólo cuenta con comandos y bloques como sentencias válidas, podría suponerse a priori dos mecanismos viables para definir dicho valor:

- previa a la ejecución de la simulación por medio de un comando, a modo de definición del STORAGE (vale la pena recordar que la mayoría de las entidades de GPSS no se definen, sino que se supone su existencia: facilidades, colas, cadenas de usuarios...);
- a partir de la generación forzada de una transacción que se ejecute antes que cualquier otra transacción y que, mediante un hipotético bloque, defina el tamaño del almacén.

La opción correcta es la primera. Lo interesante aquí es que en clase fueron los alumnos quienes propusieron ambas soluciones. Además, a la hora de decidir cuál creían que era la correcta, los alumnos respondieron de inmediato y con mucha seguridad: "se necesita un comando para inicializar el tamaño de esta entidad". En efecto, la entidad

STORAGE posee un comando homónimo que permite asociar un número de unidades a cada almacén. Además, dado que se necesita un comando, es importante encontrar el lugar dentro del código donde se ingresará dicho comando. Cuando el docente les preguntó a los alumnos dónde creían conveniente ubicar dicho comando, nuevamente los alumnos contestaron "este comando tiene que estar por sobre el primer comando START, para asegurar que el almacén (STORAGE) esté definido antes de comenzar a ejecutar la simulación". Este descubrimiento fue posible porque los alumnos conocían cómo interpreta GPSS bloques y comandos, y cómo se ejecuta la cola de comandos definidos. Además, ya estaban familiarizados con el concepto de inicialización del *termination_count* de la simulación, por medio del comando START.

Ejemplo 2: el problema de la Retry Chain

Como se explicó en el capítulo 3, todas las entidades de GPSS poseen una cadena llamada *RetryChain*, cuyo propósito es encolar todas las transacciones que esperan que el estado de la entidad en cuestión sea modificado. Es posible trazar un paralelismo entre el comportamiento de la RetryChain de GPSS y el patrón de diseño *Observer* [Gamma94] propio de la Programación Orientada a Objetos, en el cual algunos objetos permanecen "pendientes" a la espera de ciertos cambios en otros objetos. En dicho patrón, cuando los objetos *observadores* reciben una notificación que indica que se produjo un cambio en un objeto *observado*, actúan en consecuencia (por ejemplo, actualizan la información que ellos mismos poseen, envían cierta información a otro objeto, envían una actualización por la salida estándar, etc.) [Gamma94]. Con algo de ayuda del docente —a modo orientativo—, los alumnos descubrieron este paralelismo por sí solos. Pero luego, el docente presentó un ejemplo real que evidencia el problema de eficiencia relacionado con la RetryChain, en el cual puede suceder que un gran número de transacciones sea removido de esta cadena y enviado a la CEC, para luego evaluar una condición y en caso de que no se cumpla, volver a la RetryChain (capítulo 3, gestión de transacciones en la Retry Chain). Bajo esta situación, el docente indicó a los alumnos que había un problema, pero no especificó cuál. El rol de los alumnos fue precisamente detectar el inconveniente —un potencial problema de ineficiencia y baja de performance— y luego proponer una solución que evite este movimiento constante

de las transacciones entre ambas cadenas, siempre contando con las herramientas conocidas hasta ese momento. En ambos casos, los alumnos sugirieron que era conveniente aislar estas transacciones en una estructura de datos auxiliar, y gestionar el ingreso y egreso de transacciones de manera manual e independiente del resto. Ante esta respuesta, el docente les preguntó *¿y cómo debería ser esta estructura?* La respuesta inmediata de los alumnos fue: "tiene que ser una cadena, pues GPSS sólo utiliza cadenas para almacenar transacciones". En efecto, GPSS cuenta con la entidad *USERCHAIN* (cadenas de usuario), que permite al programador gestionar la planificación de transacciones manualmente, y definir criterios particulares de encolamiento y desencolamiento a partir de cualquier atributo de las transacciones. El hecho de descubrir por sí solos el potencial problema de eficiencia y la existencia necesaria de esta entidad fue posible gracias a que los alumnos conocían plenamente el mecanismo de planificación de transacciones de GPSS así como también las distintas estructuras que utiliza GPSS para alojar las transacciones durante la ejecución de los modelos.

Organización de los encuentros

A lo largo de la etapa de pruebas se realizaron 5 encuentros con los alumnos de aproximadamente 2 horas cada uno. Como se mencionó previamente, el primer encuentro consistió en la presentación de varios temas teóricos, de manera similar a aquellos que se les presenta tradicionalmente a los alumnos que cursan la materia. Durante este encuentro, el docente introdujo los conceptos —que por lo general los alumnos ya conocían por otras materias— y los relacionó superficialmente con los temas que se verían en clases posteriores. Esta primera clase finalizó con el ejemplo del modelo del sistema de clasificación de frutas, cuya "ejecución", al menos intuitiva, se buscó realizar junto con los alumnos manteniéndolos lo más participativos posible. Durante este encuentro se buscó fomentar la observación y evocar la intuición y los procesos de análisis de los alumnos, mediante la detección de los principales elementos de este sistema y la propuesta de ideas para implementar, medir y mejorar el mismo. En todo momento se buscó establecer relaciones entre conceptos de anclaje que ya poseían los alumnos y los conceptos vistos durante el encuentro del día. El establecimiento de estas relaciones tenía como objetivo integrar de manera natural los temas nuevos con

conceptos previos, buscando así generar en los alumnos ideas claras y estables, y finalmente otorgarle un mayor significado a los conceptos incorporados.

Los siguientes encuentros mantuvieron un formato similar entre sí, y se caracterizaron por una orientación más práctica que teórica —los temas prácticos eventualmente devenían en conceptos teóricos—. Al comenzar cada encuentro, se les solicitaba a los alumnos que explicasen con sus propias palabras los temas vistos en el encuentro anterior, a fin de saber cuánto recordaban realmente y, a partir de esto, realizar un repaso o reforzar determinados temas para luego continuar con los temas del día. Asimismo, este ejercicio servía como breve repaso de los temas realizado por ellos mismos. La participación del docente consistía en escuchar y eventualmente ayudar al alumno a organizar los contenidos a medida que los mencionaba, pero era responsabilidad de los alumnos guiar esta parte de la clase. En caso de no recordar algún contenido, el docente intentaba recordarles ciertos problemas de los que se había hablado en la clase anterior o mencionaba ciertos términos clave que ayudasen a los alumnos a traer a la memoria este tema. Se estima que entre 15 y 20 minutos debería ser suficiente para realizar un buen repaso; sin embargo, en la práctica este repaso tomaba por lo general entre 15 y 25 minutos. Si bien se buscaba que el repaso no sobrepasase demasiado el tiempo estimado, a fin de cubrir los temas previstos para el día, el tiempo real utilizado para estas revisiones iniciales se iba adaptando de acuerdo al intercambio entre el docente y los alumnos.

Una vez completado el repaso, el docente continuaba la clase proponiendo un problema o situación particular, a fin de exponer los nuevos conceptos correspondientes al encuentro del día. Se partía desde un problema, el cual era analizado por los alumnos con la guía del docente, a fin de buscar algún mecanismo para solucionarlo utilizando las herramientas y el conocimiento adquirido hasta ese momento. Los problemas que se presentaban requerían por lo general soluciones algo complejas, pues dichas soluciones se generaban a partir de los conceptos y las herramientas conocidas, no siempre aptos para solucionar los nuevos problemas. Esto evidenciaba la necesidad de contar con nuevos elementos, hasta ahora desconocidos, que permitiesen implementar soluciones más simples y más acordes al tipo de soluciones vistas. Sólo cuando los alumnos habían detectado los nuevos elementos, el docente hacía una explicación de los mismos en

pocos minutos para terminar de completar los conceptos. Para finalizar, y con el fin de afianzar lo aprendido, se aplicaban los nuevos conceptos con un ejemplo muy simple que se enfocaba en los temas nuevos. Estos ejemplos eran desarrollados en el pizarrón, entre el docente y los alumnos; el rol del docente aquí consistía en guiar a los alumnos a la aplicación de los conceptos del día, pero eran los alumnos quienes debían decidir cómo aplicar cada concepto nuevo sobre el ejemplo propuesto. Eventualmente, el docente debía corregir o encauzar a los alumnos cuando la solución se alejaba del objetivo del ejercicio.

Una vez que el curso ha avanzado y los alumnos han adquirido una base de conocimientos amplia en relación a los temas que se imparten en el mismo, llega el momento de integrar todas las ideas y ver cómo funcionan todas juntas en un único modelo. Para ello, durante el cuarto encuentro se presentó a los alumnos un ejercicio más amplio, cuya solución requería combinar buena parte de los temas vistos en los encuentros anteriores. La mecánica de trabajo para este tipo de ejercicio se mantuvo similar a las clases anteriores: los alumnos dirigían la solución, y el docente se limitaba a mantenerlos activos y motivados, tomar notas breves o escribir en la pizarra el código dictado por los alumnos. Eventualmente, el docente podía corregir y/o proponer alternativas a las soluciones planteadas por los alumnos, siempre y cuando las mismas requiriesen la intervención docente, ya sea porque conducían a un error o porque los alumnos se encontraban en un momento confuso en el que la ayuda del docente les permitió redefinir y retomar el problema original. También se podían introducir pequeños cambios en el ejemplo original a fin de provocar nuevos conflictos cognitivos a los alumnos, lo cual los obligaba a tomar una perspectiva más amplia y quizás una solución más genérica. Una vez completado el ejercicio en la pizarra, los alumnos debían ejecutar el mismo modelo en el simulador GPSS, y analizar allí los resultados obtenidos. Dado que el código GPSS fue desarrollado en su mayor parte sobre la pizarra, los alumnos sólo debían copiarlo y completar aquellas porciones de código que no fueron escritas por ser, o bien muy obvias, o bien repetitivas. El análisis a realizar sobre el sistema simulado abarcaba, por un lado, el estudio del sistema en sí, basado en el modelo ejecutado recientemente y, por el otro, el análisis de la ejecución del modelo, considerando todo el conocimiento adquirido previamente sobre planificación de transacciones, generación de entidades, ejecución de código GPSS, etcétera.

Finalmente, en el quinto encuentro, se presentó a los alumnos un problema más complejo, el cual debía ser resuelto exclusivamente por ellos sin ayuda del docente. El objetivo del ejercicio era observar cuánto habían avanzado y comprendido los alumnos, mediante la aplicación combinada de todos los conceptos vistos durante el curso en un problema real, e interpretar los resultados obtenidos.

Al igual que en el curso tradicional de Modelos y Simulación, para la realización del ejercicio los alumnos pueden contar con todo el material teórico y práctico que tengan a su alcance: sitios web, manual de GPSS, apuntes, libros, etc. El ejercicio debe realizarse en una PC, debe poder ejecutarse, y los alumnos deben analizar los resultados obtenidos, interpretarlos y explicarlos brevemente al docente. El tiempo límite para resolver el ejercicio es de 3 horas. Adicionalmente, una vez que la solución es desarrollada y analizada, los alumnos pueden proponer soluciones alternativas y mejoras a partir del sistema planteado en el ejercicio, siempre y cuando cuenten con tiempo dentro de las 3 horas.

El rol de GPSS Interactivo en los encuentros

La herramienta GPSS Interactivo adquirió un rol central en el proceso de enseñanza de GPSS, y fue utilizada a lo largo del curso para realizar distintas tareas con propósitos muy diversos. Durante el primer encuentro, se realizó una actividad exploratoria, en la cual los alumnos debían identificar las distintas áreas de la pantalla inicial y dilucidar cómo utilizar dicha herramienta, investigando por sí solos los diálogos y las opciones disponibles. Se esperó aquí algo de prueba y error por parte de los alumnos, generando bloques y comandos al azar, y obteniendo mensajes de error en varias ocasiones. Desde luego, esto no debía ser censurado por el docente, ya que los mensajes que la aplicación presentaba a los alumnos servían también para enriquecer la experiencia de aprendizaje. El docente podía ejemplificar sobre algún elemento particular para que los alumnos pudiesen replicarlo en otros elementos. Por ejemplo, la funcionalidad de arrastrar y soltar bloques y comandos es útil, pero no evidente a simple vista y debió ser mostrada explícitamente; otro ejemplo: algunos bloques poseen ayudas visuales pero otros no, con lo cual el docente mostraba algún diálogo particular donde estas ayudan sí existían.

A lo largo del ejercicio de descubrimiento también se utilizó esta herramienta para desarrollar algunos modelos sencillos, aplicando los bloques y comandos aprendidos recientemente. Es importante destacar que, durante esta etapa, el botón que dispara la ejecución de la simulación permaneció oculto, pues el objetivo era focalizar la atención en la interfaz de carga de bloques y comandos GPSS, y no tanto en la ejecución de simulaciones; al fin y al cabo, en esta instancia del curso, los alumnos aún desconocen qué significa exactamente ejecutar una simulación, con lo cual tampoco sabían qué resultados podían esperar de una simulación en GPSS.

En los siguientes ejercicios, a medida que se introdujeron nuevos elementos de GPSS, se utilizó esta herramienta para cargar ejemplos, identificar otros elementos del mismo grupo, explorar las distintas opciones que cada elemento ofrece, y jugar a "*qué pasaría si*" con distintas combinaciones de parámetros y entidades vinculadas a bloques y comandos. Gracias a la disposición de bloques agrupados a partir de la entidad principal con la que interactúan, se recurrió constantemente al aprendizaje por descubrimiento, en el que los alumnos conocían algún bloque de un grupo, y debían deducir el propósito de otros bloques del mismo grupo. Para ello, los alumnos se basaron en el nombre del bloque, en la descripción de los parámetros, en las ayudas contextuales y en las imágenes en los bloques que contaban con galerías visuales.

En el tercer encuentro, en el que los alumnos ya conocían cómo funcionaba el simulador tradicional de GPSS por dentro, se utilizó la función de ejecución de simulaciones simples sobre GPSS Interactivo, a fin de analizar su evolución en detalle. El docente explicó brevemente en qué consistía este nuevo simulador, remarcando que tenía un propósito meramente didáctico y que no resultaba conveniente para el trabajo profesional. Esto sirve para justificar el tiempo adicional que toma ejecutar un modelo GPSS, y para explicar la forma de analizar el resultado de la simulación, muy diferente al reporte (*report*) que se utiliza comúnmente en GPSS.

Una vez que el primer modelo se ingresa y se ejecuta en la herramienta, los alumnos deben analizar por sí mismos los elementos que se muestran en pantalla, a fin de explicar con sus propias palabras qué resultados se obtuvieron y qué información resulta relevante estudiar. Además de fomentar la observación minuciosa, este ejercicio de descubrimiento busca nuevamente poner en juego la intuición, activar la evocación y

especialmente ayudar a los procesos de relación, de análisis y de síntesis, a partir de la información que entrega el sistema, combinada con el conocimiento previamente adquirido. El objetivo aquí era otorgarle un mayor significado a los conceptos vistos, que ya dejaron de ser ideas teóricas y pasaron a ser resultados visibles, datos que pueden explorar, analizar y combinar.

El ejercicio de análisis de resultados no se limitó sólo a observar los datos y su disposición en la pantalla. Una vez que los alumnos habían descubierto y analizado las distintas áreas de la pantalla de resultados, se intentó llevar este análisis a un nivel más elevado, ahora buscando explicar cómo se habían generado esos datos. Los alumnos habían aprendido que al ejecutar una simulación con esta herramienta, se generaba una lista de capturas instantáneas de la simulación, las cuales podían luego ser exploradas de manera secuencial replicando paso a paso la ejecución del modelo. La idea era que los alumnos intenten deducir cómo funcionaba el software GPSS Interactivo por detrás, es decir, cómo había hecho esta herramienta para obtener esta secuencia de *instantáneas* del modelo en ejecución. En este ejercicio, se buscó por un lado activar la imaginación y la relación, a partir del conocimiento adquirido en este y otros cursos sobre diseño de sistemas, eficiencia, bases de datos y programación concurrente, entre otros. Por el otro lado, este ejercicio promovía el razonamiento inductivo y deductivo, así como también los procesos de análisis y de síntesis, ya que los alumnos debían tomar todo lo que sabían hasta ese momento sobre la ejecución de la simulación con GPSS, repasando el ciclo de ejecución de comandos, el funcionamiento del planificador de transacciones, el movimiento de las mismas a través de la lista de bloques y entre las cadenas, la interacción con entidades permanentes y las características del reloj de simulación, y combinando todo esto con el desarrollo de una aplicación para, finalmente, elaborar por sí mismos una metodología para generar estas instantáneas de la simulación.

A fin de incentivar estos procesos cognitivos, el docente realizó algunas preguntas a los alumnos, como por ejemplo:

- ¿Cada cuánto se tomó una nueva instantánea?
- ¿En qué momento de cada instante de simulación se tomaron las instantáneas?
- ¿Por qué tal o cual entidad mostraba estos valores?

- ¿Qué supone que sucedió con tal entidad en el futuro de su simulación?

La idea aquí no era que la respuesta sea totalmente "correcta", o sea que los alumnos indiquen con exactitud cómo funciona GPSS Interactivo por dentro. Además, desde un punto de vista tecnológico y de diseño de software, siempre es posible implementar diferentes mecanismos de recolección de información durante la ejecución de la simulación. Lo importante era promover la imaginación de los alumnos, buscando siempre que se aprovechen los conceptos teóricos y prácticos ya vistos. Ante cada propuesta, el docente debía ser capaz de analizar si el razonamiento era válido, si podía deducirse a partir de lo ya aprendido, y si tenía sentido en este contexto. Para ello, cuando los alumnos introducían sus propuestas, era importante que realizasen una correlación, aunque sea intuitiva, de su propuesta con los conceptos ya vistos relativos a sistemas de eventos discretos y a ejecución de simulaciones en GPSS. En este punto, el docente jugó un rol muy importante, ya que debía ser capaz de comprender en el momento las propuestas que los alumnos hacían, analizar el nivel de validez, descomponer la propuesta en elementos vistos durante la cursada, para luego ayudar a los alumnos a generar las asociaciones entre los diferentes temas teóricos.

Capítulo 5

Resultados alcanzados

Metodología

En esta sección se detallan cuáles fueron los resultados logrados al aplicar la nueva metodología y utilizar la herramienta GPSS Interactivo en un curso de Modelos y Simulación. Durante el dictado del curso, los audios de todas las clases fueron grabados, mientras se tomaban notas sobre la dinámica del curso, las reacciones de los alumnos, los tiempos de respuesta, etc. Los documentos de audio junto a las notas brindaron información muy valiosa, la cual, combinada con los ejercicios que realizaron los alumnos, fue vital para obtener resultados que pudieron luego compararse con el curso tradicional. Ambos documentos —audios y textos— fueron utilizados como retroalimentación durante el curso, con el objetivo de analizar lo sucedido en cada encuentro, comparar con encuentros anteriores y planificar actividades, preguntas o incluso replantear los temas del próximo encuentro.

Acerca de los alumnos

En estas pruebas participaron dos alumnos, uno de la carrera de Licenciatura en Sistemas, a quien se denominará A1, y el otro de la Licenciatura en Informática, a quien se denominará A2. Si bien ambos se caracterizan por un perfil técnico, su grado de avance en la carrera era relativamente dispar. El alumno A2, de 24 años, estaba cursando las últimas materias de la carrera al momento de tomar este curso, se encontraba planificando su tesina de grado y posee ya una experiencia laboral de al menos 3 años, en los cuales ha diseñado y desarrollado sistemas de diferente envergadura y sobre varias tecnologías. Por otro lado, el alumno A1, de 20 años, contaba con una experiencia laboral de menos de 6 meses, no había participado en desarrollos previamente, y se

encontraba finalizando el tercer año de la carrera (aproximadamente, un 50% de la misma). Estas diferencias marcaron ciertas conductas de los alumnos, acentuadas durante la primera parte del curso. La mayor experiencia del alumno A2 se hizo evidente a la hora de proponer soluciones, ya que era más metódico, pausado y organizaba sus ideas antes de expresarlas. El alumno A1 tendía a generar respuestas de manera más impulsiva, especialmente al comienzo del curso, y se guiaba mucho más por la intuición en la mayoría de sus soluciones. Este alumno, sin embargo, tenía la ventaja de haber cursado recientemente materias en las cuales muchos conceptos y problemas son similares a algunos de Modelos y Simulación (por ejemplo, Programación Concurrente o Sistemas Operativos), lo que le permitió pensar soluciones basadas en otras ya conocidas. En general, su participación en clase fue más constante comparada con la del primer alumno, que comenzó a participar más a medida que avanzaba el curso pero no tanto durante los primeros encuentros. Además, el alumno A1 parecía recordar un poco más los temas vistos durante los encuentros previos, o al menos solía basarse en ellos con más frecuencia y mencionarlos más rápidamente durante los ejercicios de repaso, aunque en ocasiones su impulso por responder de inmediato requería replantear sus ideas, ya sea porque estaban algo desorganizadas o porque no se ajustaban del todo con el marco teórico.

Más allá de la diferencia de avance en la carrera y de experiencia en el ámbito laboral, la propuesta de enseñanza pudo aplicarse sin inconvenientes con ambos alumnos. Como se verá más adelante, con el correr de los encuentros el nivel de participación de los alumnos se incrementó, se pudieron cubrir los temas previstos en menos tiempo y con una mayor profundidad, y los resultados alcanzados en los alumnos al finalizar el curso fueron muy satisfactorios.

Participación en clase

Una de las principales características de este rediseño del curso es que el docente busca en todo momento generar un muy alto nivel de participación por parte de los alumnos, pues gracias al intercambio que se produce con el docente fue posible que los alumnos generen —aunque sea de manera intuitiva— las ideas preliminares que luego, combinadas con los conceptos que el docente imparte, conformaron el nuevo

conocimiento.

Durante el primer encuentro la participación de los alumnos fue dispar. Mientras que A1 mostró mucho interés en los conceptos y ejemplos que se presentaban y participó constantemente, en especial estableciendo paralelismos con cursos que había tomado recientemente, A2 permaneció atento pero tendía a adoptar un rol pasivo, y aquí fue mucho más necesario el accionar del docente haciéndole preguntas o solicitándole que aportase ejemplos. Si bien durante este encuentro inicial se realizó un repaso de conceptos y se introdujeron algunos temas nuevos, el docente se apoyó constantemente en los conceptos ya conocidos por los alumnos para fomentar así su participación. Además, los alumnos aún no estaban familiarizados con la dinámica del curso, que es diferente a la que estaban acostumbrados en sus trayectorias académicas, en las que no suelen formar parte de la clase de manera activa y participativa. A medida que avanzaban las clases, era más natural la participación de los alumnos a partir de los problemas y ejemplos que se presentaban, pues los alumnos ya habían adquirido más conocimiento y habían comprendido la dinámica propuesta. El docente podía utilizar ejemplos ya vistos para retomar ideas o construir nuevos conceptos agregando nuevos elementos a sistemas o modelos previamente analizados. Sin embargo, si bien el nivel de participación se incrementó paulatinamente, promediando la mitad del curso aún era el docente quien debía fomentar el inicio del diálogo y las discusiones abiertas, aunque a diferencia de las primeras clases, una vez iniciada la interacción fluía de manera natural. Se observó que el mayor grado de participación en la clase se alcanzó una vez que los alumnos ya habían adquirido un nivel de conocimiento general más amplio, puesto que tenían más herramientas para participar en los debates y proponer sus propias ideas.

Tiempo de aprendizaje

Uno de los objetivos buscados en este trabajo era aprovechar al máximo los conceptos relativos a GPSS como herramienta de desarrollo, ejecución y análisis de simulaciones, y también reducir el tiempo de exposición por parte del docente sobre estos temas. En comparación con el curso tradicional de Modelos y Simulación, el tiempo dedicado al aprendizaje de GPSS fue reducido considerablemente, pasando de 7 clases, de aproximadamente 2 horas cada una, dedicadas a esta herramienta, a 4 clases en las

cuales se trató GPSS de manera exclusiva, y una clase (la primera) en la cual se vieron algunos aspectos muy generales al final de la misma, mientras se analizaba el ejemplo del ojo robótico (Anexo 1). Esta reducción de tiempo puede explicarse por la inclusión de una mayor cantidad de temas en cada clase —lo que fue motivado principalmente por los debates generados durante las clases—, permitiendo abarcar la misma cantidad de temas en un menor número de encuentros. Si bien, en teoría es de suponer que el tiempo dedicado al debate quitará tiempo al dictado de nuevos temas, en la práctica este tiempo resultó ser mejor aprovechado tanto por los alumnos como por el docente, ya que los temas discutidos quedaban mucho más fijos y en la mayoría de los casos no requirieron volver a tratarse [Acedo13]. Esto también señala que durante el curso original de Modelos y Simulación se revisan los temas ya vistos demasiadas veces, quizás porque la poca interacción docente-alumno no logra hacer tan significativo el aprendizaje, ya sea porque el docente no fomenta la participación de los alumnos, o porque estos no lo adoptan un rol activo de manera natural.

En este sentido, es importante destacar el rol que cumplió en cada una de las clases el ejercicio de repaso inicial de la clase anterior. Dado que eran los propios alumnos quienes debían explicar los temas vistos, el repaso servía no sólo para fijar los conceptos vistos, sino también como revisión de dichos temas [Acedo05][Acedo13], lo cual permitía que adquirieran el nuevo conocimiento de la clase del día de manera más natural, logrando así una clase más fluida y, nuevamente, fomentando más la participación de los alumnos en los debates y discusiones. Incluso en los casos en los que los alumnos no recordaban tanto aquello que se había tratado en la clase previa (o al menos eso creían cuando el docente se los solicitaba), el docente podía apoyarse sobre los debates generados durante la clase anterior para ayudar a los alumnos a recordar y, por lo general, la misma herramienta GPSS Interactivo servía para ayudar a los alumnos a recordar y organizar sus ideas. Dado que habían participado activamente en dichos debates, era mucho más probable que recordasen las ideas que habían planteado durante su intervención, y a partir de allí los conceptos teóricos y prácticos relacionados a dichas ideas.

El concepto aquí es bastante simple pero eficaz: si un alumno participa activamente en un debate un día, es de esperar que en la clase siguiente recuerde cuáles aspectos del

debate consideraba positivos, cuál fue su aporte, o incluso con qué partes del debate no estaba de acuerdo. En cambio, si el alumno no participa en los debates y se dedica simplemente a observar, es más probable que en la clase siguiente no recuerde tan claramente qué se debatió y qué nuevas ideas surgieron. Por este motivo, era importante que la planificación de las clases considerase la respuesta de los alumnos y los debates generados en clases anteriores, retomándolos no sólo para el repaso inicial sino también para avanzar hacia los objetivos del día, ya sea mediante el agregado de nuevos elementos o a partir del análisis de casos adicionales. Si bien el docente seguía una organización temática preestablecida, no se mantenía una estructura fija sino que se adaptaba constantemente al intercambio entre los alumnos y el docente, y al grado de aprendizaje que el docente observaba en los alumnos a partir de las respuestas que estos brindaban. En otras palabras, cada clase requería una planificación adaptada, a fin de reforzar conceptos y combinar lo sucedido en los encuentros anteriores con los objetivos propuestos para el día. Nuevamente, esto fue posible gracias al elevado grado de interacción docente-alumno, que le permitió al docente reconocer el grado de comprensión de los conceptos e identificar aquellos temas que despertaban mayor interés en los alumnos.

Aportes observados en el uso de GPSS Interactivo

Como ya se mencionó, la aplicación GPSS Interactivo fue utilizada de manera intensiva durante las primeras clases, para luego pasar paulatinamente hacia una versión profesional de GPSS (GPSS/W, desarrollada por la empresa Minuteman Software). De todos modos, la migración hacia GPSS/W no evitó que los alumnos utilicen la primera herramienta como soporte para acceder a información de bloques, comandos y entidades. Esto se observó de manera constante durante la creación de modelos propios por parte de los alumnos; en caso de tener alguna duda con alguna entidad en particular, los alumnos podían recurrir a cualquier documentación en línea o a notas personales sobre la misma; sin embargo, por lo general preferían utilizar GPSS Interactivo como primer soporte, y en caso de que la información que esta herramienta ofrece no fuera suficiente, recurrían a otros medios alternativos (por lo general, al manual de GPSS que se encuentra en el sitio web de Minuteman Software). Esta preferencia sugiere que

resultaba más práctico para los alumnos utilizar el sistema de diálogos de GPSS Interactivo que acceder a la documentación de GPSS en formato de texto. Además, al utilizar GPSS Interactivo como sistema de ayuda, se observó que en general tomaba muy pocos segundos encontrar la información buscada; en muchos casos, al visualizar los diálogos interactivos, los alumnos incluso decidían completar la información del diálogo e insertar el bloque o comandos, para luego copiar esto dentro de GPSS/W. Si bien este uso podría generar demasiada dependencia en la aplicación GPSS Interactivo, en la práctica esto no sucedió; luego de utilizar un mismo diálogo algunas veces, los alumnos recordaban más fácilmente cuáles eran los parámetros de configuración de las entidades que buscaban utilizar (esto pudo observarse tanto durante el desarrollo de los modelos GPSS en las clases, como al responder preguntas directas que hacía el docente).

La inclusión de galerías de imágenes resultó positiva especialmente durante los primeros encuentros con el lenguaje GPSS en general, y con la aplicación GPSS Interactivo en particular. Al abrirse los distintos diálogos, los alumnos tendían a acceder a las galerías (en aquellos casos que los diálogos las tuvieran), para inducir el comportamiento las entidades en la etapa de descubrimiento, o asegurarse que estaban visualizando el diálogo de una entidad correcta. Luego de los primeros encuentros, esta opción dejó de ser utilizada por los alumnos, puesto que el conocimiento adquirido les permitía obtener toda la información sobre una entidad a partir del grupo al que pertenecía y de los parámetros de configuración que requería. Más allá de esto, las imágenes utilizadas en las galerías fueron evocadas constantemente por los alumnos a la hora de plantear sugerencias o proponer ejemplos en los debates en clase. Esto indica que dichas imágenes no sólo sirvieron para ayudar a fijar información sobre las entidades, sino que también fueron aprovechadas por los alumnos como soporte para construir sus propias ideas.

La separación de bloques y comandos aportó un nivel de claridad que ayudó desde el primer momento a organizar estas dos entidades centrales de código de GPSS. Desde el comienzo del curso, los alumnos observaban cómo se distribuían los comandos en la zona superior del código, y los bloques en el área inferior. Cuando comenzaron a utilizar GPSS/W, los alumnos replicaban automáticamente esta organización (separando los bloques y los comandos en dos espacios bien diferenciados); además, cada vez que

utilizaban un bloque que requería la definición de una entidad mediante un comando, generaban ambas partes del código (comando y bloques) todo junto, lo cual evitó que surgieran errores posteriores durante la compilación y/o ejecución de la simulación. Fue también positiva la visualización de los comandos como una cola, en la cual el orden de declaración de los mismos es un factor crucial que determinará la ejecución de la simulación. Al usar GPSS/W, los alumnos se aseguraban siempre de mantener organizada esta cola de comandos, dejando comandos *de ejecución* como START, CLEAR y RESET al final de la cola, y comandos de definición e inicialización como STORAGE, VARIABLE, TABLE y EQU al principio de la cola. Es importante destacar aquí que GPSS Interactivo no implementa la totalidad de los comandos; de hecho, sólo implementa un comando de ejecución (START) y pocos comandos de definición (FUNCTION, STORAGE, EQU). Sin embargo, la clasificación en dos categorías y su organización dentro de la cola de comandos surgió de manera natural con estos pocos comandos, y fue escalada a la totalidad de los comandos cuando fue necesario sin presentar ningún inconveniente.

En el mismo sentido, la capacidad de GPSS Interactivo de organización del código mediante arrastrar y soltar (*drag&drop*) permitió que los alumnos visualicen cualquier sentencia de GPSS como un bloque completo, o sea como una entidad que incluye a los parámetros, el nombre de la entidad, la etiqueta y hasta la posición en la lista o cola, según sea la entidad. Por ejemplo, los alumnos creaban primero los bloques que deseaban utilizar, y luego las disponían en la lista de bloques, en los lugares que correspondía en cada caso. Esto sugiere que no estaban pensando al programa GPSS que estaban escribiendo como una lista de sentencias que se ejecuta secuencialmente, sino que lo trataban como una red de entidades interconectadas, en la que cada entidad debe ocupar un lugar específico; de aquí puede inferirse que los alumnos no estaban pensando en líneas de código, sino en las formas de interacción con entidades, a través de los bloques y los comandos.

El agrupamiento de bloques según la entidad principal con la que interactúan fue un buen aporte durante la etapa de descubrimiento, ya que los alumnos lograron inferir el significado de buena parte de los bloques a partir de su nombre y del grupo al que pertenecían. Esta característica también fue muy útil durante el desarrollo de modelos, ya que permitió que los alumnos encontrasen rápidamente los bloques buscados al

momento de construir sus modelos. Durante una actividad en una de las primeras clases, por ejemplo, el alumno sabía que tenía que trabajar con una entidad que sirviese para representar a un peluquero en un ejemplo típico conocido como "el barbero dormilón" [Hartley98]. Cuando el docente le pregunta al alumno qué entidad funciona bajo este mecanismo (atendiendo un cliente por vez, y haciendo que el resto espere su turno), el alumno se dio cuenta de inmediato de que necesitaba una facilidad; esto lo llevó directamente al grupo de bloques para facilidades, lo cual le permitió insertar los bloques SEIZE y RELEASE sin mayores inconvenientes. Cabe destacar que, en este punto del curso, el alumno sólo sabía de la existencia de las facilidades y estos bloques, pero las había visto muy poco en funcionamiento.

Durante la fase de análisis de la ejecución de la simulación, una de las primeras observaciones realizadas indica que la pantalla que muestra los resultados de la simulación resultó muy intuitiva y fácil de comprender para los alumnos, ya que no presentaron inconvenientes a la hora de navegar por las distintas entidades e interpretar los datos que se presentaban en pantalla. Una de las funciones más utilizadas por los alumnos fue la navegación secuencial basada en el reloj de simulación, ya que buscaban seguir paso a paso la ejecución del modelo, al principio con un alto grado de observación pero luego de una manera más superficial, hasta que se encontraban con algún evento en particular que estaban buscando (la aparición de una transacción, el cambio de estado de una facilidad, etcétera). Durante la navegación secuencial, los alumnos prestaban asimismo mucha atención al valor del *termination count* de la simulación, lo que les permitía observar cómo se acercaba al final de la simulación a medida que se acercaba a cero, y que esto sucedía de forma totalmente independiente al avance del reloj de simulación o al movimiento de las transacciones entre las cadenas del sistema. Con esto, se logró dar relevancia a este atributo, vital para comprender el funcionamiento del motor de simulación pero muchas veces confundido con otros elementos de GPSS, como el reloj de simulación o el atributo *generation limit* del bloque GENERATE.

En general, los alumnos observaron minuciosamente la información presentada en pantalla sobre facilidades y almacenes. Sin embargo, no se observó en ningún caso que los alumnos prestasen demasiada atención a la información detallada sobre las entidades en general y sobre las transacciones en particular. Si bien buena parte de esta

información resultaba irrelevante a esta altura del curso (principalmente conceptos como *assembly set*, *block count*, *trace indicator* y *delay indicator*, entre otros), es probable que la forma de presentar los datos no invitase a los alumnos a analizar en detalle esta información. De todos modos, es importante destacar que en general los alumnos sí se concentraron en observar algunos atributos más relevantes y significativos para esta etapa del aprendizaje (en especial *transaction ID*, *block departure time*, *next secuential block* y *priority*) al observar las transacciones en las distintas cadenas. Estas observaciones indican que podría rediseñarse la vista de las transacciones, a fin de destacar sólo información útil y colocando como información el resto de los atributos. Una alternativa interesante aquí consiste en incorporar funciones de adaptabilidad al software, que le permitan al docente seleccionar ciertos atributos para destacar según la etapa de aprendizaje en la que se encuentre el curso, o según las necesidades de los alumnos. Por ejemplo, si se está trabajando con el modo SIM del bloque TRANSFER, el atributo *delay indicator* de las transacciones cobra mayor relevancia, y es de esperar que los alumnos busquen dicho atributo al analizar en detalle el estado de una transacción. De este modo, el sistema servirá de guía tanto para el alumno como para el docente, marcando las distintas etapas del aprendizaje y resaltando información importante según sea la etapa en la que se encuentre.

Aplicación de los conocimientos

Uno de los parámetros más significativos a la hora de evaluar el nivel de aprendizaje está dado por la capacidad de aplicar el conocimiento adquirido en una situación real [Barriga94]. Por tal motivo, en el quinto encuentro se les presentó a los alumnos un problema concreto, el cual involucraba un sistema que debían modelar, representar y ejecutar en GPSS y, finalmente, analizar e interpretar los resultados de esta ejecución. Durante este ejercicio se aplicaron las mismas reglas de evaluación que se utilizan durante las evaluaciones en el curso tradicional de Modelos y Simulación, con lo cual fue posible establecer comparaciones y paralelismos entre ambos cursos. Las reglas de evaluación acotan el tiempo de resolución del ejercicio a una duración máxima de 3 horas. Si bien la evaluación es individual, los alumnos pueden acceder a cualquier documentación que deseen (libros, apuntes personales, sitios web); también pueden

utilizar cualquier herramienta informática como soporte (GPSS/W, planillas de cálculo, editores de texto), y pueden realizar consultas al docente sólo si se trata de dudas relacionadas con el enunciado entregado; las preguntas relacionadas con temas teóricos o prácticos vistos en clase no son respondidas por el docente. Adicionalmente, para este ejercicio se permitió que los alumnos dispusiesen de GPSS Interactivo como otra herramienta informática de soporte.

Los ejercicios fueron resueltos por los alumnos en sus propias computadoras personales. Allí desarrollaron los modelos en GPSS, los ejecutaron (mediante GPSS/W), y tomaron nota de los resultados. Durante la resolución del ejercicio, utilizaron mucho GPSS Interactivo como guía, aunque los modelos fueron desarrollados principalmente en GPSS/W. En pocas ocasiones requirieron acceder a la documentación en línea de GPSS, en particular para revisar ciertas entidades que no figuran en GPSS Interactivo. Una vez finalizado el ejercicio, el docente les hizo algunas preguntas sobre los resultados obtenidos, a fin de evaluar si realmente habían comprendido lo que había sucedido durante la ejecución de sus modelos.

En todos los casos, los alumnos pudieron resolver los ejercicios sin mayores problemas; si bien en ciertos casos decidieron representar algunas entidades reales con diferentes entidades de GPSS, los modelos seguían siendo válidos ya que se respetaba el funcionamiento del sistema propuesto y era posible obtener estadísticas similares, necesarias para analizar los resultados más adelante. Uno de los aspectos más interesantes de las soluciones propuestas fue que desde el principio, los alumnos las diseñaron pensando en los datos que debían recolectar y que iban a necesitar analizar más adelante. Esto significa que no se concentraron en simplemente hacer funcionar el modelo, sino que se aseguraron de contar con las entidades apropiadas para recolectar datos y obtener estadísticas sobre tiempos de espera, niveles de ocupación, tiempos de tránsito, etcétera. Esto puede explicarse porque los alumnos conocían bien las entidades más comunes que iban a necesitar para construir su modelo, con lo cual pudieron concentrarse desde un principio en cuestiones más avanzadas de sus modelos. Cabe destacar que, por lo general, se ha observado que los alumnos buscan armar un "esqueleto" simple del modelo que funcione, y posteriormente amplían dicho modelo incorporando entidades más complejas y entidades para recolectar datos. Esta

metodología es más fácil de manejar para ellos, pues comienzan con algo simple que paulatinamente se va haciendo más complejo, pero tiene un inconveniente no menor: en caso de que el primer modelo no esté correctamente planteado pero, aun así, "funciona" (en el sentido de que puede ser ejecutado en GPSS sin errores), los alumnos sólo notarán el error una vez que han incorporado aquellas entidades que recolectan ciertos datos que evidencian tales errores. Esto los lleva muchas veces a volver a pensar y desarrollar el modelo original desde el principio. Cuando esto sucede, por lo general dedican más tiempo a tratar de construir un modelo que funcione según el sistema original y, como consecuencia, muchas veces no alcanzan las 3 horas para completar el modelo, analizar los resultados obtenidos, y generar una documentación mínima de los mismos.

Cuando los alumnos tuvieron que analizar sus modelos y responder preguntas, no tuvieron ningún inconveniente en explicar el funcionamiento de los modelos, el significado de los resultados obtenidos, y el porqué de dichos resultados. Si bien en todos los casos los alumnos se limitaron a recolectar y analizar la cantidad mínima de datos (esto es, ningún alumno realizó un modelo más amplio pensando en un diseño más abstracto o en futuras ampliaciones), el alumno A2 detectó, en base a las preguntas que el docente le hacía sobre los resultados obtenidos en su modelo, que el sistema original propuesto en este ejercicio poseía ciertos problemas de performance. Además, modificó algunos datos de entrada del modelo y lo ejecutó reiteradamente con pequeños cambios, lo cual le permitió encontrar una combinación de parámetros de entrada que optimizasen el sistema subyacente. Esta solución alternativa sugiere que el alumno adquirió la capacidad de realizar experimentos sencillos e intuitivos sobre el modelo, en los que a partir de la combinación de diferentes valores de parámetros de entrada y el análisis intuitivo de los valores de la salida, podía modelar soluciones más eficientes para aplicar al sistema original.

Un tiempo después

Una vez finalizado el ejercicio de evaluación, se comunicó a los alumnos que había algunos temas de GPSS que aun no habían sido tratados, y que si lo deseaban podían ser vistos más adelante. Ambos respondieron afirmativamente, con lo cual se pautó un nuevo encuentro con fecha a confirmar. Pasados 45 días del último encuentro, el docente

convocó a los alumnos a este nuevo encuentro con el objetivo de completar los temas faltantes. Sin embargo, este encuentro buscaba también evaluar cuánto recordaban de todos los temas vistos en los encuentros anteriores. Cabe destacar que durante estos 45 días no se realizaron encuentros, actividades ni repasos. Además, al comenzar este último encuentro, se les preguntó a los alumnos si habían repasado algo y, como era de esperar, se obtuvo una respuesta negativa en todos los casos.

En líneas generales, los alumnos recordaban muy bien todo lo visto. Al principio presentaban ideas algo sueltas o dispersas, pero con algo de ayuda del docente pudieron organizarlas sin mayores problemas. Se observó que el alumno A1 recordaba muchos detalles, en particular relacionados con los debates y ejemplos, y que se basaba en ellos para organizar sus pensamientos. El alumno A2 declaró al comenzar el encuentro que no recordaba casi nada de los conceptos vistos, pero luego de escuchar algunas ideas generales pudo reconstruir sin mayores inconvenientes buena parte de los temas tratados. Si bien los alumnos comenzaron en todos los casos por temas más generales, el docente les hacía determinadas preguntas que los llevaban a analizar aspectos más específicos y complejos, que pudieron ser explicados en todos los casos. A diferencia del repaso que se hacía en todos los encuentros, este ejercicio de repaso duró en promedio 1 hora, lo cual representa más del doble de tiempo estimado para este ejercicio. La extensión de tiempo puede explicarse porque los conceptos no eran tan recientes para los alumnos, con lo cual tenían que evocar la memoria de largo plazo para reconstruir las ideas y conceptos aprendidos. Además, y más importante aún, durante este repaso se trataron muchos más temas que en un repaso típico, que suele abarcar sólo lo visto en el encuentro anterior.

Una vez completado el repaso, comenzó la introducción del tema nuevo, el cual consistía en la entidad GROUP (grupo), junto con sus bloques asociados. Si bien los grupos pueden ser tanto de transacciones como de números, el foco principal se puso sobre grupos de transacciones.

A diferencia de los casos anteriores, los bloques para interactuar con los grupos son bastante más complejos, pues poseen una mayor cantidad de parámetros de configuración, y algunos de estos parámetros pueden ser más complejos o “rebuscados”, comparados con los típicos parámetros de los bloques que los alumnos conocían hasta

ahora. Por ejemplo, el bloque SCAN analiza todos los elementos (transacciones) de un grupo hasta encontrar uno que cumpla cierta condición; al hacerlo, se guarda en un atributo de la transacción activa (la que ejecutó el SCAN) algún valor de la transacción miembro del grupo para la cual la condición evaluada resultó verdadera. Desde luego, todo esto requiere un operador condicional (para evaluar la condición) junto a una gran cantidad de parámetros de configuración, que incluye (entre otros) el atributo de los miembros del grupo a evaluar, el valor contra el cual dicho atributo será comparado, el atributo a copiar una vez que la evaluación resultó satisfactoria, y el atributo en la transacción activa donde se guardará dicho valor.

En general, los alumnos se sintieron cómodos con los grupos de transacciones, ya que esta entidad les ofrece un mecanismo para trabajar con muchas transacciones a la vez, es más simple que los conjuntos de ensamble o *Assembly Set* (usados principalmente para sincronizar transacciones), y no bloquea el normal flujo de las mismas a lo largo de la lista de bloques (como sí sucede con las cadenas de usuario o USERCHAIN). Sin embargo, algunos de los nuevos bloques (principalmente SCAN, EXAMINE y ALTER) presentaban un grado mayor de complejidad al que estaban acostumbrados. De todos modos, el mecanismo de aprendizaje aplicado fue el mismo que se utilizó durante los encuentros anteriores: los alumnos debían proponer el funcionamiento de los bloques, en base a lo que conocían previamente. Por ejemplo, siguiendo con el caso del bloque SCAN, el docente les indicó a los alumnos que el objetivo de este bloque era detectar alguna transacción de un grupo que cumpla alguna condición particular. Con esta información tan breve, los alumnos propusieron algunas ideas sobre cómo funcionaría por dentro el bloque, y de allí qué parámetros mínimos serían requeridos para asegurar este funcionamiento.

En este bloque surgen algunas características nuevas, propias de otros bloques de grupos, como el uso de un atributo de la transacción activa para almacenar un valor, y la referencia a un atributo de alguna transacción pasiva (miembro del grupo) desde el cual tomar dicho valor. Por ejemplo, dada la siguiente sentencia:

```
...
2 SCAN E Lote10,NumeroParte,932,Precio,PrecioParte,LlamarPorTelefono
...
```

se realizará una búsqueda dentro del grupo Lote10 (parámetro A) hasta encontrar la primer transacción cuyo atributo NumeroParte (parámetro B) tenga el valor 932 (parámetro C, operador E por *equal*). Al encontrarla, se copiará el valor del atributo Precio de dicha transacción (parámetro D) en el atributo PrecioParte de la transacción activa (parámetro E). En caso de no encontrarse ninguna transacción dentro del grupo que cumpla dicha condición, la transacción activa será enviada al bloque etiquetado como LlamarPorTelefono (en el cual se simulará, por ejemplo, que se realiza un pedido para reponer el stock de la parte 932 en el Lote10).

Al presentarse este bloque, pero sin conocerlo en detalle, los alumnos comenzaron a deducir cuáles eran los parámetros mínimos que iban a necesitar para poder configurarlo correctamente. Algunos parámetros surgieron de inmediato, como por ejemplo los parámetros A, B, C. El problema nuevo al que se enfrentaron los alumnos era cómo obtener un valor de una transacción del grupo, y principalmente dónde colocar dicho valor para luego utilizarlo. Las alternativas que se evaluaron incluyeron la creación de una entidad específica para esto, el uso de alguna entidad global preexistente (en particular, un SAVEVALUE), y el uso de un atributo propio de la transacción activa. Lo interesante aquí es que los dos alumnos descartaron de inmediato la opción 1, y entre las opciones 2 y 3, prefirieron la 3, ya que en ningún caso GPSS almacenaba valores globales por sí solo. En efecto, la alternativa 3 es la única opción correcta.

Gracias a lo aprendido con el bloque SCAN, al tratar con el bloque ALTER (de funcionamiento similar a SCAN, pero algo más complejo puesto que se altera un parámetro de las transacciones del grupo que cumplen una condición), no se encontraron mayores inconvenientes y los alumnos pudieron comprenderlo en poco tiempo.

Una vez vistos estos bloques que trabajan con conjuntos de transacciones, el docente planteó a los alumnos un último problema que surge del modo en que el motor de GPSS ejecuta estos bloques por dentro. El problema consiste en analizar qué sucede si, mientras la transacción activa analiza un grupo de transacciones evaluando algún atributo interno de las mismas, una de dichas transacciones es alterada antes o después de ser evaluada. Por ejemplo, retomando el ejemplo anterior:

...

2 **SCAN E** *Lote10,NumeroParte,932,Precio,PrecioParte,LlamarPorTelefono*

...

Si la transacción 2 pertenecía al grupo Lote10, y su atributo NumeroParte valía 100, pero durante la ejecución del bloque SCAN, dicho atributo se alteraba y pasaba a valer 932, ¿qué sucede si dicha transacción ya había sido evaluada? ¿y qué sucede si aun no había sido evaluada? ¿debía tomarse su valor original al comenzar la ejecución del bloque SCAN, o debía considerarse el valor al momento de la evaluación? Desde luego, en el problema anterior se plantea una situación que, en realidad, no tiene sentido en el contexto de GPSS. No existe posibilidad de que se modifique un atributo de una transacción del grupo mientras se está ejecutando el bloque SCAN, porque para que esto suceda debería haber otra transacción activa que la modifica (quizás mediante un ALTER), o la misma transacción 2 debería estar activa y modificar si atributo *NumeroParte* mediante un bloque ASSIGN. Pero esto no puede suceder, dado que en GPSS sólo existe una transacción activa en todo momento; si se está ejecutando el bloque SCAN, entonces la transacción activa será aquella que ingresó a dicho bloque, y ninguna otra podrá ejecutarse hasta que esta transacción se detenga.

Al analizar la situación, el alumno A1 comenzó a buscar una solución desde el punto de vista transaccional de las bases de datos, a fin de asegurar una ejecución atómica [Silberchatz91] de la operación SCAN. Si bien este razonamiento es válido desde el punto de vista funcional, conceptualmente —en el contexto de GPSS— es una idea incorrecta. De todos modos, al escuchar la propuesta, el docente incentivó al alumno a describir su idea con mayor detalle, para ver si podía implementarse de ese modo dentro del motor de simulación de GPSS. Cuando el alumno empezó a buscar dónde convenía colocar una marca, como si fuera un semáforo, para bloquear otras transacciones, se dio cuenta de inmediato de que no podía darse la situación de otra transacción intentando ejecutarse, con lo cual descartó de plano la solución atómica. Por otro lado, el alumno A2 propuso como primera solución que no era necesario preocuparse por esto, dado que al haber una sola transacción activa en todo momento, se aseguraba de antemano la atomicidad de la operación. Si bien el primer alumno había propuesto una solución inválida al principio, pudo corregirse por sí solo y llegó a la misma conclusión que el segundo

alumno, lo cual indica que ambos habían comprendido bien este aspecto del funcionamiento del motor de simulación de GPSS.

Conclusiones

A lo largo de este capítulo se explicó cómo la organización temática del curso de Modelos y Simulación, la actitud frente a la clase tanto del docente como de los alumnos, y el aprovechamiento de una herramienta pensada exclusivamente para el dictado del curso, influyeron tanto en la forma de comprender como en el tiempo requerido para aprender una tecnología relativamente compleja como GPSS. En estos párrafos se enumeraron las nuevas propuestas del curso de Modelos y Simulación, las cuales incluyen la incorporación de la herramienta de trabajo GPSS Interactivo, combinado con una reorganización temática de la primera parte del curso de Modelos y Simulación que aprovecha gran parte de los aportes de dicha herramienta como soporte para la enseñanza. GPSS Interactivo fue utilizado como vehículo para fomentar la participación activa de los alumnos en la clase, para estimular mediante la exploración el autodescubrimiento de muchos de los conceptos tratados en clase, y para promover una forma distinta de pensar el modelado de sistemas para simulación, orientado principalmente hacia la recolección de datos y los tipos de análisis que se realizarán sobre los mismos.

Esta propuesta fue puesta en práctica con dos alumnos avanzados de las carreras de Licenciatura en Sistemas y Licenciatura en Informática de la Facultad de Informática de la Universidad Nacional de La Plata (Argentina). Esto representa entre un 40% y 50% de la cantidad usual de alumnos que asisten presencialmente a las clases del curso tradicional. Estos alumnos participaron de 5 encuentros de aproximadamente 2 horas cada uno, en los cuales se les impartieron la gran mayoría de los conceptos que se ven en el curso de Modelos y Simulación relativos a sistemas, modelos y simulación en general, y al simulador GPSS en particular.

Durante la aplicación de esta propuesta, se observó que los alumnos infirieron y comprendieron por sí mismos parte de los nuevos conceptos, a partir de conceptos anteriores y con algo de guía del docente. Esto permitió aprovechar mejor el tiempo utilizado para el dictado del curso, puesto que se requerían menos repasos o revisiones

de temas anteriores y, a medida que avanzaba el curso, los alumnos eran capaces de generar y asimilar conceptos más complejos. Como consecuencia de esto, el tiempo total requerido para aprender los mismos temas fue reducido cerca de un 50%, y se obtuvieron también resultados aceptables en las evaluaciones finales. Más importante aún, los alumnos demostraron que el conocimiento adquirido permanecía vigente 45 días después de finalizados los encuentros, en un encuentro adicional realizado para completar los temas de GPSS, pero también para conocer cuánto recordaban de todo lo visto durante el curso.

Capítulo 6

Conclusiones y trabajos futuros

La simulación es una técnica que está enormemente difundida en la ciencia; desde la física hasta la biología, pasando por la economía y la arquitectura, en todos estos campos juega un papel importante [Pugnaloni08]. El uso apropiado de la simulación como herramienta de análisis de sistemas y predicción del comportamiento futuro requiere un profundo conocimiento de muchas áreas de la ciencia y una capacidad para integrar esta gran diversidad de áreas temáticas. En este sentido, conceptos de matemática y estadística, abstracción y modelado, programación, minería de datos y diseño de experimentos convergen cada vez que se desarrolla un modelo de simulación. Existen muchos tipos de modelos de simulación diferentes, que se adaptan a las características del sistema que modelan y a los objetivos del estudio para el cual se han realizado. Uno de estos tipos de modelos es el que resulta útil para la simulación de sistemas de eventos discretos, comúnmente conocidos como DES por sus siglas en inglés (Discrete Event System).

La gran diversidad de áreas de la ciencia que abarca la simulación impone un gran desafío para los estudiantes que comienzan a interiorizarse en esta técnica, y que pueden ser de áreas muy diversas como las ciencias informáticas, las distintas ramas de la ingeniería, las ciencias naturales o las ciencias económicas. Al momento de tomar un curso de simulación, los estudiantes deben contar con muchos conceptos previos, y en algunos casos deben haber adquirido un profundo conocimiento de los mismos. Pero,

como indicó I. Ståhl [Ståhl00], esto no siempre sucede dado que por lo general los cursos previos —en especial, aquellos de estadística— abarcan muchos temas pero no lo hacen con la suficiente profundidad. En algunos casos, el problema se hace aún más complejo, ya que los alumnos no cuentan siquiera con las nociones más básicas requeridas; esto sucede, por ejemplo, cuando se trata de alumnos que no poseen un curso de introducción a la programación en sus carreras: ciencias económicas y ciencias naturales son casos típicos donde esto sucede. Sin embargo, si bien en otras carreras los alumnos adquieren conocimientos de programación, aquí deben enfrentarse a nuevos paradigmas de programación muy distintos a los que conocen. Los lenguajes de programación específicos para realizar simulaciones en general, y GPSS en particular, no siguen las convenciones de otros lenguajes de programación de paradigmas como la programación procedural y la programación orientada a objetos. Aquí los programas carecen de una estructura modular clara, las sentencias condicionales y de iteración no existen o existen como efectos secundarios de otras sentencias, los conceptos típicos de constantes, variables y funciones son muy diferentes, y, finalmente, cada línea de código encapsula de manera transparente la invocación de subrutinas, la interacción entre muchas entidades y la recolección de datos para futuros análisis estadísticos. Como consecuencia de estas diferencias y disparidades, recae en el rol del docente la habilidad para adaptar los contenidos del curso, la metodología de enseñanza a aplicar en cada clase, y las actividades que se realizan con los alumnos, a fin de poder enseñar a los estudiantes los conceptos de simulación en un tiempo relativamente acotado.

La simulación se apoya fuertemente en las herramientas que se utilizan para generar modelos, ejecutarlos, obtener resultados y analizarlos. Con el avance en la capacidad de procesamiento y almacenamiento de las computadoras, estas herramientas se han vuelto más avanzadas, los lenguajes de simulación son cada vez más abstractos, los entornos incorporan más y más funciones, y se han implementado diversos mecanismos de integración que permite aplicar un mismo paradigma o una misma herramienta de simulación a contextos muy diferentes.

Si bien los constantes avances en entornos de desarrollo, lenguajes de programación y herramientas de análisis repercuten de manera positiva en el ámbito profesional, el académico y el científico, es evidente que las herramientas profesionales requieren una

formación cada vez mayor por parte de los usuarios si se desea aprovecharlas en todo su potencial. Estas herramientas no fueron diseñadas para la etapa de aprendizaje inicial, y su uso en clase no resulta adecuado cuando se está trabajando con estudiantes que aún no han adquirido —o están comenzando a hacerlo— los conceptos mínimos relativos a simulación de eventos discretos, dado que el docente requerirá dedicar mucho tiempo del curso al aprendizaje de una herramienta en particular.

Estos problemas han sido enfrentado por muchos docentes e investigadores, y durante los últimos 20 años se han generado diferentes aproximaciones. Por un lado, algunos investigadores se han dedicado a estudiar y proponer un marco común para el diseño de cursos de simulación [Garcia09][Centeno09][Ståhl00], en los cuales las herramientas que se utilizan cumplen un rol secundario y el foco principal es puesto en los elementos que conforman el curso, las características de los alumnos que participan del mismo y los objetivos con los que se planificó el curso. Otros investigadores han puesto el énfasis en el diseño y desarrollo de herramientas dedicadas a la enseñanza de la simulación, en particular de GPSS. Estas herramientas en general buscan lidiar con uno o más aspectos propios del aprendizaje de GPSS, como el nivel de abstracción y falta de modularidad de GPSS, la complejidad del motor de simulación, o en análisis de los resultados de la simulación. Las propuestas abarcan desde librerías y lenguajes de programación que toman elementos de GPSS y los combinan con elementos de lenguajes de programación procedurales [Zikic96][Ueno80][Barnett82], hasta entornos de aprendizaje que permiten construir modelos mediante diálogos, gráficos y una fuerte interacción entre el estudiante y las partes del modelo mediante arrastrar y soltar [Fonseca09][Herper99]. También se ha propuesto la integración del desarrollo y ejecución de modelos en GPSS con el entorno de gestión de enseñanza conocido como Moodle [Despotović09].

Muchos autores se han concentrado principalmente en el diseño de los cursos, dejando las herramientas de software utilizadas para el aprendizaje como un elemento más del curso. Por otro lado, las distintas implementaciones de GPSS orientadas a la enseñanza proponen mejoras que buscan simplificar algún aspecto de este lenguaje de manera aislada. En estos trabajos, sin embargo, se ha tenido en cuenta el diseño del curso, y las herramientas propuestas no poseen elementos que sirvan para acompañar el

dictado de los distintos conceptos teóricos y su traducción en conceptos prácticos. Adicionalmente, no se han considerado mecanismos de asistencia a los alumnos, mediante sistemas de consejos contextuales, íconos, ayudas embebidas o enlaces a bibliografía complementaria. En este sentido, vale la pena destacar los trabajos de algunos investigadores en el área del diseño de entornos de aprendizaje interactivos, mecanismos de búsqueda de ayuda, desarrollo de sistemas de asistencia y ayuda, y diseño de sistemas tutores [Aleven03][Anderson95][Aleven00][Aleven01][Wood99] [Renkl02][Puustinen98].

Desde el punto de vista de la disponibilidad, a excepción de JGPSS, todos los desarrollos propuestos poseen licencias de software privativas, lo cual no permite el estudio del motor de simulación por dentro, y la introducción de cambios y mejoras como parte de las actividades del curso. Además, estas aplicaciones requieren un pago por cada copia del software que se utiliza; si bien existen algunos descuentos para su uso dentro de ámbitos académicos, asumir tales costos se hace difícil para instituciones de bajos recursos, o para instituciones públicas gratuitas en las cuales muchos de los alumnos no pueden costear un gasto de estas características. Esto impone una barrera económica adicional, que se aparta de las políticas de inclusión social y educación igualitaria que promueven tantas instituciones públicas y gobiernos.

Las tecnologías actuales que dan soporte a Internet han realizado incontables avances. Los navegadores web se han vuelto verdaderos entornos de ejecución de aplicaciones, los lenguajes de programación web tanto del lado del cliente como del lado del servidor permiten una flexibilidad comparable a la que ofrecen las tecnologías de escritorio, y la velocidad y ubicuidad de las conexiones a Internet permiten acceder a aplicaciones web desde casi cualquier lugar del planeta, utilizando una cada vez mayor variedad de dispositivos. Entre las herramientas propuestas, sólo FonWebGPSS fue diseñada exclusivamente para funcionar sobre Internet, integrándose con el entorno de gestión de enseñanza conocido como Moodle. Sin embargo, vale la pena destacar que esta aplicación no aporta elementos para el aprendizaje de GPSS, sino que ofrece un área de texto para introducir código GPSS, el cual es luego ejecutado en un servidor que enviará finalmente los resultados de la simulación para ser mostrado mediante gráficos y tablas. Otra aplicación WebGPSS, propone el uso de GPSS a través del navegador

mediante una *applet* de Java. Además de basarse en una tecnología ya en desuso, esta aplicación es, en realidad, una versión de WinGPSS adaptada para su funcionamiento dentro de un *applet*. Por lo tanto, puede observarse que no existen implementaciones de GPSS que aprovechen las ventajas de las aplicaciones web, y que sean realmente multiplataforma, o sea que puedan ejecutarse independientemente del sistema operativo subyacente. WinGPSS posee versiones para GNU/Linux y para OSX (Mac), pero estos desarrollos ya han sido discontinuados. Tampoco existen aplicaciones comerciales de GPSS con soporte multiplataforma; si bien tanto GPSS/H como GPSS World pueden ser ejecutadas en sistemas GNU/Linux a través de aplicaciones como Wine[19] o Crossover[20], esto introduce una importante pérdida de performance y genera un punto adicional de inestabilidad sobre estas aplicaciones.

En esta tesis, entonces, se introdujo una herramienta de software interactiva diseñada específicamente para el aprendizaje de GPSS en el contexto de un curso de simulación. Esta herramienta, denominada GPSS Interactivo, fue concebida teniendo en cuenta muchas de las recomendaciones de los investigadores que estudiaron el diseño de sistemas de enseñanza interactivos, y para su implementación se han tomado elementos de muchas de las herramientas analizadas. También se han introducido nuevas características, y se ha puesto el foco en el uso de la misma a lo largo de un curso de simulación, en particular durante los primeros encuentros en los cuales se imparten muchos conceptos básicos e iniciales (sistema, modelos, simulación, etcétera) y se aprende a utilizar el lenguaje GPSS.

GPSS Interactivo se compone principalmente de dos aplicaciones. Una de ellas contiene el motor de simulación de GPSS, se ha desarrollado en lenguaje de programación Java y utiliza la base de datos MySQL. El uso de estas tecnologías abiertas permite su ejecución sobre cualquier sistema operativo, dada la independencia de la plataforma de las aplicaciones Java, y la disponibilidad del motor de bases de datos MySQL para todos los sistemas operativos. Si bien esta aplicación fue pensada para ejecutarse en un servidor remoto, es también viable que los alumnos descarguen este motor de simulación y lo ejecuten en sus propias computadoras. Para simplificar esto, se

[19] WineHQ - Ejecuta aplicaciones de Windows en Linux, BSD, Solaris y Mac OSX. Página web: http://www.winehq.org/.

[20] CrossOver Runs Windows on Mac and Linux, easily and affordably. Página web: http://www.codeweavers.com/products/.

ha incorporado la herramienta de gestión de proyectos de software conocida como Apache Maven[21], que se encarga de la compilación y la descarga de las dependencias, librerías y paquetes necesarios para asegurar la correcta ejecución de este motor de GPSS. Si bien también se ha desarrollado un entorno de trabajo interactivo, cualquier programador puede interactuar con este simulador mediante una serie de funciones que permiten enviar código GPSS para su ejecución, conocer el avance de la ejecución y el tiempo restante estimado, y recuperar el estado completo de la simulación en cualquier instante de la ejecución. Esto es posible ya que este motor de simulación almacena en la base de datos el estado y las relaciones de todas las entidades que conforman la simulación, lo cual conforma un historial completo de la simulación que sirve de apoyo para comprender mejor el funcionamiento interno del motor de simulación.

La otra herramienta que forma parte de GPSS Interactivo es un entorno de aprendizaje interactivo diseñado para permitir a los estudiantes construir modelos GPSS utilizando diálogos, apoyados por ayudas contextuales, galerías de imágenes, y principalmente una disposición de elementos en pantalla que permite fijar conceptos muy importantes de GPSS de manera intuitiva y desde el primer momento que se enfrentan a este lenguaje. Este entorno interactivo fue desarrollado con tecnologías web que se ejecutan dentro del navegador web (HTML5, CSS3 y Javascript, principalmente), lo cual asegura una completa portabilidad a cualquier plataforma capaz de ejecutar un navegador web moderno. Para ejecutar modelos, esta herramienta web interactúa con el motor de simulación a través de una aplicación web intermedia, también desarrollada en Java, que traduce los distintos eventos del usuario en requerimientos HTTP, los cuales son enviados al simulador y cuya respuesta es devuelta al cliente por medio de mensajes AJAX con formato JSON. Además, la ejecución de este entorno interactivo es totalmente independiente del motor de simulación que se utilice, con lo cual es posible construir los modelos GPSS sin tener que acceder al servidor web. Por ser una aplicación web, tampoco se requiere de un proceso de instalación de software.

El entorno interactivo de GPSS fue diseñado y desarrollado tomando en consideración los distintos conceptos que se ven en el curso de simulación, las actividades que se realizan con los alumnos y los ejemplos con los que se trabaja en clase. Por este motivo,

[21] Apache Maven Project, página web: http://maven.apache.org/.

fue posible integrar fácilmente esta herramienta durante el curso de Modelos y Simulación, acompañando las distintas actividades propuestas en clase y permitiendo traducir conceptos teóricos en ejemplos prácticos y viceversa. Durante el diseño, se han tenido en consideración muchas de las propuestas relativas al desarrollo de ILE y a los sistemas de asistencia y ayuda en los mismos. También se han tomado algunos elementos de otros desarrollos, como la interacción con bloques y comandos mediante eventos como arrastrar y soltar, el uso de diálogos para la configuración de los mismos, y la agrupación de los distintos bloques según el tipo de entidad con la que interactúan o la función que cumplen (característica sólo observada en JGPSS).

La implementación de GPSS Interactivo fue puesta a prueba en el curso de Modelos y Simulación de la Facultad de Informática de la Universidad Nacional de La Plata. Para ello, se diagramó el curso para hacer uso de las facilidades que esta herramienta introducía. Se modificó el orden de los temas dictados con el objetivo de incorporar desde el primer momento los conceptos más avanzados relativos al funcionamiento del motor de simulación de GPSS y de las características generales de sus bloques, comandos y entidades. Luego, a medida que avanzaban los encuentros, se estudiaron en profundidad los detalles más específicos de las distintas entidades y sus aportes en los modelos de simulación. También se fomentó en todo momento la participación de los alumnos en el curso, por medio de debates y actividades conjuntas con el docente aprovechando varias características de diseño de esta herramienta. El objetivo fue mantener un alto interés de los alumnos en los encuentros, fomentar el descubrimiento y aprendizaje por sí mismos, y traducir en todo momento los conceptos teóricos aprendidos en ejemplos concretos y ejercicios en clase. El uso de GPSS Interactivo sirvió para fomentar la observación y evocar la intuición y los procesos de análisis, mediante la detección de los principales elementos de este sistema y la propuesta de ideas para implementar, medir y mejorar el mismo. Se establecieron relaciones entre conceptos de anclaje que ya poseían los alumnos y los conceptos vistos durante el encuentro del día, con el objetivo de generar en los estudiantes ideas claras y estables, y otorgarles un mayor significado a los conceptos incorporados.

Los resultados obtenidos durante cada encuentro, y al finalizar el curso, son muy alentadores. Se observó que los alumnos fueron capaces de inferir y comprender por sí

solos parte de los nuevos conceptos, a partir de conceptos anteriores y con algo de guía del docente. Esto permitió aprovechar mejor el tiempo utilizado para el dictado del curso, puesto que se requerían menos repasos o revisiones de temas anteriores, y a medida que avanzaba el curso, los alumnos eran capaces de generar y asimilar conceptos cada vez más complejos y abstractos por sí mismos. Como consecuencia de esto, el tiempo total requerido para aprender los mismos temas que se dictan en el curso tradicional de Modelos y Simulación fue reducido cerca de un 50%, y se obtuvieron también muy buenos resultados en las evaluaciones finales. Los alumnos demostraron que el conocimiento adquirido permanecía vigente 45 aún días después, en un encuentro adicional realizado para completar algunos temas adicionales de GPSS, pero también para conocer cuánto recordaban de todo lo visto durante el curso. Durante este encuentro final, los alumnos fueron capaces de recordar no sólo los principales conceptos vistos durante los encuentros anteriores, sino también un muy alto nivel de detalle en cuanto a determinados aspectos específicos de algunas entidades o del funcionamiento del motor de simulación de GPSS.

El uso de GPSS Interactivo durante los encuentros, y la introducción de esta nueva propuesta de enseñanza, requiere mucha flexibilidad por parte del docente, quien debe observar constantemente las reacciones, el grado de participación y las respuestas de los alumnos, y ajustar consecuentemente el rumbo de cada encuentro. En el contexto de esta tesis, las pruebas se realizaron con un grupo acotado de sólo 2 alumnos, lo cual representa cerca de la mitad de los alumnos que suelen participar de los encuentros del curso de Modelos y Simulación de la Facultad de Informática de la UNLP. Si bien los resultados obtenidos fueron muy buenos, debe discutirse la aplicación de esta misma metodología en un curso con un número mayor de alumnos. El docente debe seguir muy de cerca a cada estudiante, debe saber en todos los encuentros qué recuerda, qué ha comprendido y qué le ha parecido más interesante, y debe ser capaz de adaptarse en consecuencia. Esto podría realizarse con un grupo más amplio, de 5 o incluso hasta 10 alumnos, pero es evidente que resulta cada vez más inviable a medida que aumenta la cantidad de participantes.

Cabe destacar que una de las principales características de la aplicación GPSS Interactivo es su capacidad para almacenar todo el historial de cada simulación, el cual

luego puede ser recuperado por partes y analizar detenidamente cómo ha avanzado la ejecución de los modelos. Esto tiene sentido, desde luego, para modelos pequeños tanto en el número de bloques y comandos como en la cantidad de entidades que se generan. Si bien esto ha resultado muy ventajoso, el almacenamiento de esta gran cantidad de información —del orden de varios miles de entidades— requiere mucho tiempo en comparación con el tiempo requerido para ejecutar un modelo. Se han incorporado así mecanismos para que el almacenamiento se realice en segundo plano y de manera concurrente, a fin de permitir un análisis inicial de resultados de la simulación una vez completada, y visualizar el detalle de la ejecución una vez que ha finalizado su almacenamiento. De todos modos, es importante destacar que mientras que la ejecución de un modelo pequeño puede tomar entre 2 y 3 segundos, su almacenamiento puede demorar hasta 60 segundos. Esto genera una demora bastante extensa en el contexto del curso, en especial cuando se están realizando pruebas concretas para estudiar algún aspecto específico del motor de simulación. Por lo tanto, es preciso mejorar este aspecto de GPSS Interactivo si se desea hacer un uso más intensivo, especialmente si se considera que podrá ser utilizado por más alumnos al aplicarse en cursos con mayor concurrencia.

Por otra parte, una de las características de JGPSS es su diseño pensado para ser ampliado por los estudiantes, mediante la implementación de ciertos módulos o clases que conforman el motor de simulación desarrollado en Java. El uso de este lenguaje simplifica considerablemente la incorporación de código de los alumnos, dado que por su perfil de programadores ya conocen tanto el lenguaje como las herramientas de desarrollo necesarias para utilizarlo. Los autores de JGPSS afirman que esto permite a los alumnos comprender en mayor profundidad el funcionamiento del motor de simulación de GPSS, ya que no sólo estudian sus aspectos teóricos y visualizan su funcionamiento a partir de los resultados obtenidos en los modelos, sino que desarrollan su propio motor de simulación. Esta idea puede ser aplicada en el contexto de esta tesis, dado que el motor de simulación implementado también se ha realizado en Java, y su código se encuentra disponible de manera libre bajo una licencia GPLv2. Los alumnos pueden obtener una copia del software, estudiar su estructura y diseño, y realizar todo tipo de modificaciones. Sin embargo, mientras que esta herramienta fue pensada para ser extendida con nuevas entidades, bloques y comandos, no fue diseñada considerando

modificaciones en el funcionamiento interno del simulador. Por el lado de JGPSS, su diseño considera la integración de nuevos elementos y modificaciones en áreas específicas del motor de simulación, y además los autores han generado guías de codificación para los estudiantes, esqueletos vacíos con comentarios para introducir código, y otros elementos que sirven para orientar a los alumnos para abrir y modificar el motor de simulación. Este es un aspecto muy interesante, y por lo tanto es necesario analizar la posibilidad de implementar mejoras en el diseño del motor de simulación así como también nuevas herramientas y elementos que promuevan y simplifiquen la extensión del motor de simulación.

El uso de entornos de aprendizaje a distancia o LMS (Learning Management Software), como Moodle, Sakai[22] y Dokeos[23], está cada vez más difundido en muchos contextos de enseñanza. En el caso particular de la UNLP, se ha creado la Dirección de Educación a Distancia[24] (EAD), como una herramienta complementaria para la formación de sus alumnos de grado y posgrado, que integra esfuerzos previos de las diferentes facultades de la UNLP. En este sentido, se ha elaborado un plan que contiene entre sus ejes el desarrollo de tecnología que cree condiciones y que posibilite acercar contextos o facilitar tareas y recursos para que otras personas desarrollen su proceso de aprendizaje, la capacitación de sus docentes en el empleo de herramientas de EAD, el diseño de experiencias multidisciplinarias y la definición de metodologías y normas que aseguren la calidad de los cursos no presenciales que se imparten desde la universidad [EAD13]. La universidad cuenta, a su vez, con diversas herramientas web como el campus virtual Aula Cavila[25], el entorno virtual WebUNLP[26] y la herramienta de Blogs de Cátedra[27].

La integración de las herramientas de aprendizaje con los entornos de enseñanza presenta muchas ventajas, muchas de las cuales son aprovechadas por la aplicación FONWebGPSS (Imagen 53). Esta aplicación introduce, por un lado, un módulo para

[22] Sakai Project | collaboration and learning - for educators by educators. Página web: http://www.sakaiproject.org/.
[23] Dokeos | Open Source E-Learning. Página web: http://dokeos.com/.
[24] Dirección de Educación a Distancia de la Universidad Nacional de La Plata, página web: http://www.unlp.edu.ar/educacionadistancia.
[25] Campus Virtual Latinoamericano - Universidad Nacional de La Plata, página web: http://www.cavila.unlp.edu.ar/.
[26] WebUNLP - Entorno Virtual de Enseñanza y Aprendizaje, página web: http://webunlp.unlp.edu.ar/
[27] Blogs de Cátedras de la UNLP, página web: http://blogs.unlp.edu.ar/

docentes, que les permite crear, insertar, probar y gestionar casos y problemas de simulación de eventos discretos, y también evaluar el trabajo de los estudiantes y realizar reportes; por otro lado, FONWebGPSS posee un módulo para estudiantes, quienes pueden acceder a las actividades propuestas por el docente y crear a partir de allí sus propios modelos, realizar ajustes en el entorno gráfico, y analizar los resultados de la simulación. Los alumnos pueden también realizar la presentación de sus modelos y reportes al docente a través de esta herramienta. Si bien no se incluyen elementos pedagógicos específicos para el desarrollo de modelos y análisis de resultados, la integración entre la herramienta de trabajo y el LMS resulta muy positiva para el desarrollo del curso en general.

Imagen 53: Integración de FONWebGPSS con Moodle [Despotović09]

Al igual que cualquier aplicación web, GPSS Interactivo puede ser integrado en un entorno web de manera fácil, por ejemplo mediante una etiqueta IFRAME o con muy poco esfuerzo de programación del lado del servidor. Sin embargo, esta integración será sólo desde el aspecto visual y no desde lo funcional. La flexibilidad en el diseño del curso que permite FONWebGPSS, así como en la creación y evaluación de actividades, y en el trabajo con varios alumnos a la vez hacen que deba considerarse seriamente la

integración de GPSS Interactivo con algún LMS, o mejor aún, el desarrollo de una capa de software especial que permita la integración de esta herramienta con cualquier software web. Esto también permitirá contar fácilmente con usuarios registrados, una capacidad inexistente en GPSS Interactivo en este momento, junto a las ventajas de registrar sus actividades y los resultados de sus evaluaciones. El uso intensivo de tecnologías como AJAX y JSON en el desarrollo de la interfaz de usuario de GPSS Interactivo hacen que esto sea muy viable, pero es preciso generar una API que brinde cierta organización y estabilidad cuando se invoquen componentes de esta aplicación desde sistemas de terceros.

Las entidades de GPSS poseen muchos atributos propios que el programador debe conocer y saber utilizar para hacer uso de las mismas de manera correcta. Además, las transacciones son un tipo de entidad que permiten al programador sumarle otros atributos adicionales. En el contexto de un curso de simulación, especialmente durante las primeras instancias de ese curso, es discutible la necesidad de mostrar la totalidad de los atributos internos de cada una de las entidades, dado que si bien permite obtener una visión completa de las entidades, esto carece de utilidad puesto que el estudiante aún desconoce el significado de la gran mayoría de ellos. Sin embargo, GPSS Interactivo muestra, al analizar la ejecución de cada simulación, todos los atributos que conforman cada una de las entidades y, según se ha observado durante los encuentros, esto tiende a confundir a los estudiantes, en especial en las primeras instancias del curso. Además, algunos atributos cobran mayor sentido cuando se están tratando determinados temas muy específicos de GPSS. Por ejemplo, las cadenas PendingChain e InterruptChain de las Facilidades cobran sentido cuando los alumnos comprenden las particularidades del bloque PREEMPT, y el atributo AssemblySet de las transacciones sólo es útil al momento de aprender los mecanismos de sincronización de transacciones y los bloques asociados (SPLIT; ASSEMBLE; GATHER; MATCH). Deberá considerarse por lo tanto un rediseño de la vista de las entidades en el reporte final, a fin de destacar sólo la información útil y colocando como información el resto de los atributos. Una alternativa interesante aquí consiste en incorporar funciones de adaptabilidad al software, que le permitan al docente seleccionar ciertos atributos para destacar según la etapa de aprendizaje en la que se encuentre el curso. Esto puede también beneficiarse con la integración de GPSS Interactivo y otros LMS, puesto que el docente podría configurar la vista de las entidades

de GPSS en el mismo momento en que carga las actividades y los materiales de estudio. Como consecuencia, el docente podría organizar el curso de manera más simple y coherente, asegurándose de que la herramienta de trabajo se adapte con cada nuevo tema y, a la vez, que al introducir los conceptos teóricos los alumnos contarán con la asistencia virtual necesaria para aplicarlos.

Además de mostrar y ocultar atributos de las entidades según la instancia en la que se encuentra el curso, la visualización del modelo puede también enriquecerse a través de vistas específicas de entidades. La implementación actual permite visualizar el modelo como un todo, y avanzar o retroceder el reloj de simulación a fin de visualizar los cambios en las distintas entidades durante la ejecución.

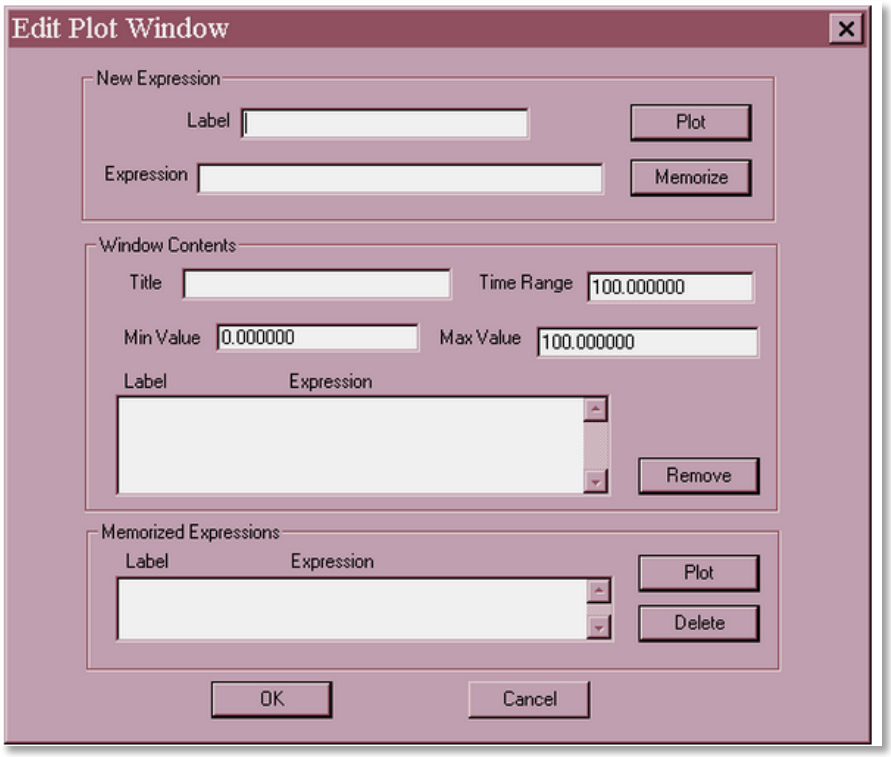


Imagen 54: Ventana de edición de expresiones y de generación de gráficos de GPSS World

Sumado a la vista global del modelo, sería interesante sumar un mecanismo de navegación por entidades, que permita por un lado acceder exclusivamente a la vista de una única entidad en un momento dado, incluyendo otras entidades con las que se relaciona en ese mismo momento, y por otro lado analizar el progreso de la simulación visualizando el modelo desde el interior de esta entidad. Por ejemplo, para el caso de las

facilidades, es posible generar una vista integral que muestre cómo ha variado el porcentaje de ocupación, la cantidad de transacciones que la han tomado, o la longitud de cada una de sus cadenas; por el lado de las transacciones, podría crearse una visualización que permita observar desde qué instante y hasta qué instante de simulación "existe" dicha transacción, con qué entidades ha interactuado en cada instante, y cómo se han modificado sus atributos internos. Es importante destacar que esto es posible de hacerse, al menos parcialmente, con la herramienta GPSS World mediante el uso de expresiones dentro de la ventana de gráficos (*plot window*, Imagen 54), los cuales son mostrados al finalizar la simulación (Imagen 55). Sin embargo, GPSS World asume que el usuario conoce cómo generar estas expresiones, lo que requiere estar familiarizado con los SNA disponibles para cada entidad, con la sintaxis utilizada para generar estas expresiones, y con el uso de esta ventana de gráficos.

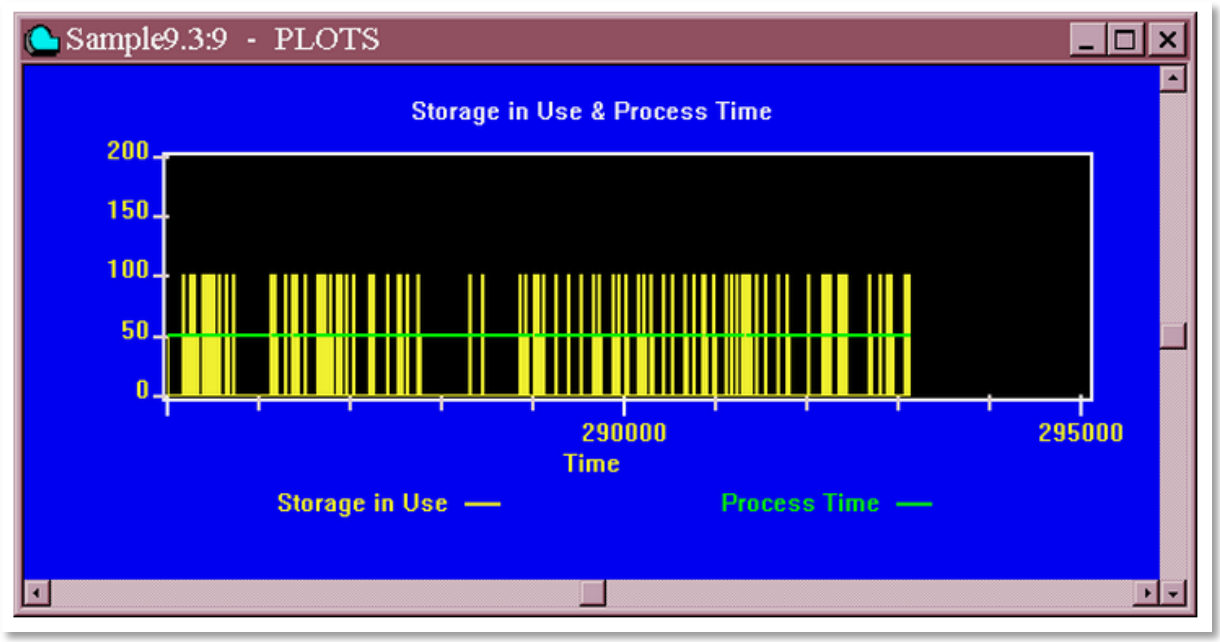


Imagen 55: Visualización de un gráfico que combina el nivel de uso de un storage con el tiempo de procesamiento de una facilidad, en GPSS World

Capítulo 7

Referencias

[Aaby96] Aaby, A. A. (1996). *Introduction to programming languages*.

[Acedo05] María de Lourdes Acedo de Bueno. "Formación docente para promover una visión constructivista en el diseño de cursos presenciales a través de la elaboración de planes y programas". *Revista Comportamiento*, vol. 7 no. 1, 2005, p. 93-112.

[Acedo13] María de Lourdes Acedo de Bueno. "El aprendizaje significativo en la docencia". En línea. http://buenoacedodocentes.homestead.com/aprendizajesignificativo.html.

[Adelsberger86] Adelsberger, H. H., Pooch, U. W., Shannon, R. E., Williams, G. N. "Rule-based Object Oriented Simulation Systems", *Proc. SCS Conf. Intelligent Simulation Environments*, San Diego, Ca., Jan. 1986, p. 107-112.

[Aleven00] Aleven V. and Koedinger, K. R. "Limitations of student control: Do students know when they need help?" In *Proceedings of the 5th International Conference of Intelligent Tutoring Systems, ITS* 2000, p. 292-303.

[Aleven01] Aleven V. "Helping students to become better help seekers: Towards supporting metacognitioin in a Cognitive Tutor". *1st NSF-DFG Workshop on Learning Technologies*, Tübingen, Germany. 2001.

[Aleven01A]Aleven V. and Koedinger, K. R. "Investigations into help seeking and learning with a Cognitive Tutor". In *Papers of the AIED-2001 Workshop on Help Provision and Help Seeking in Interactive Learning Environments*. Brighton, U.K., 2001.

[Aleven03] V. Aleven, E. Stahl, S. Schworm, F. Fischer, R. Wallace. "Help Seeking and Help Design in Interactive Learning Environments". *Review of Educational Research*, Fall 2003. Vol. 73, no. 3, 2003, p. 277-320.

[Anderson84] Gordon E. Anderson, "The coordinated use of five performance evaluation

methodologies", *Communications of the ACM*. Vol. 27 no. 2, Feb.1984, p. 119-125.

[Anderson05] Anderson J. R., Corbett, A. T., Koedinger, K. R. and Pelletier, R. "Cognitive Tuttors: lessons learned". *Journal of the Learning Sciences*. Vol. 4, 1995, p. 167-207.

[Arias03] J. J. Pazos Arias, A. Suárez González, R. P. Díaz Redondo. *Teoría de Colas y Simulación de Eventos Discretos*. Pearson-Prentice Hall. Universidad de Vigo. 2003.

[Arrollo00] Arrollo, I., Beck, J.E., Woolf, B.P., Beal, C.R., and Schultz, K. "Macro-adapting Animalwatch to gender and cognitve differences with respect to hint interactivity and symbolism". In *Proceeding of the 5th International Conference on Intelligent Tutoring Systems, ITS 2000*. Berlin, 2000, p. 574-583.

[Arrollo01] Arrollo, I. Beck, J.E., Beal, C.R., Wing, R. and Woolf, B.P. "Analyzing students' response to help provision in an elementary mathematics intelligent tutoring system". In *Papers of the AIED-2001 Workshop on Help Provision and Help Seeking in Interactive Learning Environments.* Ed. R. Lucking. 2001

[Banks82] Stephen D. Roberts, Jerry Banks, James Kho, Udo Pooch, and John Ramberg. "Teaching simulation to undergraduates". In *Proceedings of the 14th conference on Winter Simulation. Vol. 2, 1982*, p. 706-706.

[Banks84] J. Banks, J.S. Carson, B. L. Nelson, D.M. Nicol. *Discrete-Event System Simulation*. Prentice-Hall. 1984.

[Barnett82] C. C. Barnett, "Micro Passim: A Discrete-Event Simulation Package for Microcomputers Using UCSD Pascal". In L.A. Leventhal, *Modelling and Simulation on Micromputers*. The Society of Computer Simulation, La Jolla, Californa. 1982.

[Barriga04] Ángel Díaz Barriga. "Una polémica en relación al examen". *Revista Iberoamericana de Educación - Calidad de la Educación*. No. 5, Agosto 1994.

[Biles87] Biles W.E. "Introduction to simulation". *Proc. 1987 Winter Simulation Conference*, p. 7-14.

[Bobillier76] P. A. Bobillier, Basil Charles Kahan, A. R. Probst. *Simulation with GPSS and GPSS V*. Prentice-Hall. 1976.

[Busse99] Busse, S., Kutsche, R. D., Leser, U., & Weber, H. *Federated information systems: Concepts, terminology and architectures.* Technische Universität Berlin, Fachbereich 13-Informatik. 1999.

[Centeno01] Centeno, M. A., M. Carillo. "Challenges of introducing simulation as decision

making tool". *Proceedings of the 2001 Winter Simulation Conference*. Piscataway, New Jersey, USA. 2009.

[Chen02] Gilbert Chen and Boleslaw K. Szymanski. "Lookahead, Rollback and Lookback: Searching for Parallelism in Discrete Event Simulation". *Proc. Summer Computer Simulation Conference*. San Diego, CA, July 2002.

[Conway86] Richard Conway, William L. Maxwell, "XCELL: a cellular, graphical factory modelling system", *Proceedings of the 18th conference on Winter simulation*, December 08-10, 1986, Washington, D.C., United States, p. 160-163

[Cooper02] Jim Cooper. *Using MS-DOS 6.22.* 3ra edición. ISBN 0789725738, 9780789725738. Editor Que Publishing, 2002.

[Crain97] Robert C. Crain. 1997. "Simulation using GPSS/H". In *Proceedings of the 29th conference on Winter simulation (WSC '97)*, Sigrún Andradóttir, Kevin J. Healy, David H. Withers, and Barry L. Nelson (Eds.). IEEE Computer Society, Washington, DC, USA, p. 567-573.

[Crain99] R. C. Crain, J. O. Henriksen. "Simulation using GPSS/H". In *Proceedings of the 1999 Winter Simulation Conference*. Phoenix, AZ, USA. 1999

[Dahlman08] Dahlman, Parkvall, Skold and Beming. *3G Evolution*, 2nd Edition. HSPA and LTE for Mobile Broadband. 2008. Academic Press. ISBN: 9780123745385

[DeGiusti08] De Giusti, Marisa Raquel; Lira, Ariel Jorge; Villarreal, Gonzalo. "Simulation Framework for teaching in Modeling and Simulation Areas". *European Journal of Engineering Education*, vol. 33, issue 5 & 6, Oct. 2008 , p. 587-596.

[Despotović09] Marijana S. Despotović, Božidar Lj. Radenković, Dušan M. Barać: "GPSS for e-learning Environment". *International Conference on Telecommunications in Modern Satellite, Cable and Broadcasting Services - TELSIKS 2009,* Serbia, Niš, p. 318-321.

[DeVargas06] Elaine de Vargas. "La situación de enseñanza y aprendizaje como sistema de actividad: el alumno, el espacio de interacción y el profesor". *Revista Iberoamericana de Educación*, ISSN-e 1681-5653, vol. 39, no. 4, 2006.

[Díaz03]Díaz Barriga, F: "Cognición situada y estrategias para el aprendizaje significativo", en *Revista Electrónica de Investigación Educativa*, vol. 5, no. 2, 2003.

[Dillon98] Dillon, A. and Gabbard, R. "Hypermedia as an educational technology: A review of the quantitative research literature on learnine compehension, control and style". *Review*

of Educational Research, vol. 68, 1998, p. 322-349

[Dorwarth97] Dorwarth, H., P. Lorentz, K. C. Ritter, T, J. Schriber. "Towards a Simulation and Animation Environment for the Web". Winter Simulation Conference WSC 1997.

[Dunna06] Eduardo García Dunna, H. García Reyes, L. E. Cárdenas Barrón. *Simulación y análisis de sistemas con ProModel*. Pearson Educación, 2006.

[EAD13] Historia de Educación a Distancia de la UNLP. En línea http://www.unlp.edu.ar/historiaead . Último acceso: mayo 2013.

[Farnsworth86] Kenneth D. Farnsworth, Van B. Norman, T. A. Norman. "Integrated software for manufacturing simulation", *Proceedings of the 18th conference on Winter simulation*, December 08-10, 1986, Washington, D.C., United States, p. 184-192.

[Fonseca09] Fonseca i Casas, P.; Casanovas, C.J., "JGPSS, An open source GPSS framework to teach simulation," Simulation Conference (WSC), *Proceedings of the 2009 Winter*, 13-16 Dec. 2009, p. 256-267

[Ford87] Donnie R. Ford, Bernard J. Schroer, "An expert manufacturing simulation system", *Simulation*, vol. 48 no. 5, May 1, 1987, p. 193-200.

[Gamma94] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994. ISBN 0201633612, p. 293-304.

[García09] H. García and M. A. Centeno. "S.U.C.C.E.S.S.F.U.L.L.: a framework for designing discrete event simulation courses". *Proceedings of the 2009 Winter Simulation Conference*. Austin, TX, 2009. p. 289-298.

[Garrett05] Garrett, Jesse James, *et al.* "Ajax: A new approach to web applications". 2005. Disponible en http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications

[Gómez12] Peña Gómez, D. . *Diseño de un framework de persistencia.* Universitat Oberta de Catalunya. 2012.

[Gordon86] Robert F. Gordon , Edward A. MacNair, Peter D. Welch, Kurtiss J. Gordon, James F. Kurose, "Examples of using the RESearch Queueing Package Modeling Environment (RESQME)", *Proceedings of the 18th conference on Winter simulation*, December 08-10, 1986, Washington, D.C., United States, p. 494-503.

[Hartley98] Stephen J. Hartley. *Concurrent Programming. The Java Programming Language*. Oxford University PRESS. 1998. ISBN 978-0195113150.

[Herick03] Marjan Hericko, Matjaz B. Juric, Ivan Rozman, Simon Beloglavec, and Ales Zivkovic. 2003. "Object serialization analysis and comparison in Java and .NET. SIGPLAN", *no.* 38, 8 August 2003, p. 44-54.

[Herper99] Henry Herper, Ingolf Ståhl, "Micro-GPSS on the Web and for Windows: a tool for introduction to simulation in high schools", *Proceedings of the 31st conference on Winter simulation: Simulation a bridge to the future*, December 05-08, 1999, Phoenix, Arizona, United States, p. 298-306.

[Herper03] Herper, Henry and Ståhl, Ingolf. "Modeling and simulation in high school education - two European examples". Simulation Conference, 2003. *Proceedings of the 2003 Winter.* Inst. for Simulation & Graphics, Otto-von-Guericke Univ., Magdeburg, Germany, vol. 2, p. 1973-1981, 2003.

[Herscovitch65] Herscovitch, H.; Schneider, T. H., "GPSS III — An expanded general purpose simulator," *IBM Systems Journal* , vol. 4, no. 3, p. 174-183, 1965.

[Hopper98] James W Hopper, "Strategy-related characteristics of discrete-event languages and models", *Simulation*, vol. 46 no. 4, p. 153-159, April 1986.

[IEEE00] "High Level Architecture (HLA) - Framework and Rules". *IEEE Standard for Modeling and Simulation.* 1516-2000, pp. i,22, 2000.

[Jamsa02] Kris A. Jamsa: "Hacker Proof: The Ultimate Guide to Network Security. General Interest". 2nd edition, p. 384-400. *Cengage Learning*. ISBN 0766862712.

[Karian98] Zaven A. Karian, Edward J. Dudewicz. *Modern Statistical, Systems, and GPSS Simulation*, Second Edition. CRC Press. 1998.

[Kachitvicyanukul90] Voratas Kachitvicyanukul, James O. Henriksen, Richard E. Nance, C. Dennis Pegden, Charles R. Standrige, Brian W. Unger, "Simulation environment of the 1990's (panel)", *Proceedings of the 19th conference on Winter simulation*, December 14-16, 1987, Atlanta, Georgia, United States, p. 455-460.

[Khoshnevis87] Khoshnevis, B., Chen, A. "An Automated Simulation Modeling System Based on AI Techniques", *Proceedings of SCS Conference AI and Simulation*, San Diego, CA, 1987, p. 87-91.

[Kleine71] Kleine H. "Second Survey of Users View of Discrete Simulation Languages". In *Simulation*, vol. 17 issue 2, p. 89-& , 1971.

[Klein97] Klein U. and S. Straßburger. "The High Level Architecture: Requirements of

interoperable and reusable simulations by example of traffic and infraestructure simulations". *Proceedings of the 11th Simulation Symposium ASIM 97*, Dortmund, Germany. 1997.

[Klein98] Ulrich Klein and Steffen Straßburger and Jürgen Beikirch. "Distributed Simulation with JavaGPSS Based on the High Level Architecture". In *Proceedings of the 1998 SCS International Conference on Web-Based Modeling and Simulation,* p. 85-90, 1998.

[Law00] A. M. Law, W. D. Kelton. *Simulation Modeling and Analysis*, 3th Edition. Editorial Mc Graw Hill. 2000.

[Leach05] Leach, P. J., Mealling, M., & Salz, R.. "A universally unique identifier (uuid) urn namespace", 2005.

[Lee02] Lee, Y. H., Cho, M. K., Kim, S. J., & Kim, Y. B. "Supply chain simulation with discrete–continuous combined modeling". *Computers & Industrial Engineering*, vol. 43, no. 1, p. 375-392, 2002.

[Li95] Li, W. X. "A simple and efficient incremental LL (1) parsing". In *SOFSEM'95: Theory and Practice of Informatics*. Springer Berlin Heidelberg, p. 399-404, 1995

[Lorenz97] Lorenz, P. and K. C. Ritter. Skopeo: "Platform Independent System Animation for the W3". In *Proceedings of the Simulation and Animation Conference*, Magderbur, p. 12-23, March 1997

[Low99] Low YH; Lim CC; Cai WT. "Survey of languages and runtime libraries for parallel discrete-event simulation". *Simulation*, vol. 72 Issue 3 p. 170-186.

[Malloy86] Brian Malloy, Mary Lou Soffa, "SIMCAL: the merger of Simula and Pascal", *Proceedings of the 18th conference on Winter simulation*, p. 397-403, December 08-10, 1986, Washington, D.C., United States.

[McHaney96] McHaney, R. "Simulation project success and failure: Some survey findings". Working paper, Dept. of Management, Kansas State University, Manhatan, 1996.

[Merkik05] Mernik, M., Heering, J., & Sloane, A. M.. "When and how to develop domain-specific languages". *ACM computing surveys (CSUR),* vol. 37, no. 4, p. 316-344, 2005.

[Minuteman13] Minuteman Software. GPSS Entities. Disponible en línea http://www.minutemansoftware.com/reference/r4.htm. Último acceso: mayo 2013.

[Mihran74] Mihran, G.A., *et al.,*. "What Makes a Simulation Credible?" *Proc. 5th Ann. Pfttsurgh Conf,* 1974.

[Mitchell76] Mitchell, E. E., & Gauthier, J. S. "Advanced continuous simulation language (ACSL)". *Simulation*, vol. 26, no. 3, 1976, p. 72-78.

[Möller04] Björn Möller, Lennart Olsson, "Practical Experiences from HLA 1.3 to HLA IEEE 1516 Interoperability", 04f-siw-045, www.sisostds.org, 2004.

[Murray88] Murray, K.J., Sheppard, S.V. "Knowledge-Based Simulation Model Specification", *Simulation*, vol. 50, no. 3, 1988, p. 112-119.

[Nance81] Richard E. Nance, The time and state relationships in simulation modeling, *Communications of the ACM*, vol. 24 no. 4, April 1981, p. 173-179.

[Nance84] Richard E. Nance, "Model development revisited", *Proceedings of the 16th conference on Winter simulation,* December 28-30, 1984, Dallas, TX, p. 74-80.

[Neelamkavil87] Francis Neelamkavil, *Computer simulation and modelling*, John Wiley & Sons, Inc., New York, NY, 1987.

[Okeefe86] Robert M. O'Keefe , Ruth M. Davies, "Discrete visual simulation with Pascal_SIM", *Proceedings of the 18th conference on Winter simulation*, December 08-10, 1986, Washington, D.C., United States, p. 517-525.

[Palmero04] H. García and M. A. Centeno. "S.U.C.C.E.S.S.F.U.L.L.: a framework for designing discrete event simulation courses". *Proceedings of the 2009 Winter Simulation Conference*. Austin, TX, 2009. p. 289-298.

[Park88] S. K. Park and K. W. Miller. 1988. "Random number generators: good ones are hard to find". *Commun. ACM* vol. 31, no. 10, October 1988, p. 1192-1200.

[Parr07] Parr, T. "The Definitive ANTLR Reference: Building Domain-Specific Languages (Pragmatic Programmers)". *Pragmatic Bookshelf*, May 2007.

[Pegden79] Claude Dennis Pegden, A. Alan, B. Pritsker. "SLAM: simulation language for alternative modeling". *Simulation,* vol. 33 no. 5, November 1979, p.145-157.

[Pegden95] C. Dennis Pegden, Randall P. Sadowski, and Robert E. Shannon. *Introduction to Simulation Using SIMAN* (2nd ed.). McGraw-Hill, Inc., New York, NY, USA. 1995.

[Pollacia89] Lissa F. Pollacia. 1989. "A survey of discrete event simulation and state-of-the-art discrete event languages". *SIGSIM Simul.* Dig. Vol. 20, no. 3, September 1989, p. 8-25.

[Pons10] Calvo Pons, M. A. . Frameworks de persistència: un estudi comparatiu. 2010.

[PritsKer99] A. Alan B. Pritsker,Jean J. O'Reilly. *Simulation with Visual SLAM and AweSim.* Ed. John Wiley & Sons. 1999.

[Pugnaloni08] L. A. Pugnaloni. Los simuladores. El papel de la simulación en la ciencia. En *Ciencia Hoy*, no. 105, 2008. Disponible en http://www.iflysib.unlp.edu.ar/?q=node/100

[Puustinen98] Puustinen, M. "Help-seeking behavior in a problem-solving situation: Development of self-regulation". *European Journal of Psychology of Education*, vol. 13, p. 271-282. 1998.

[Renkl02] Renkl, A. "Learning from worked-out examples: Instructional explanations supplement self-explanations". *Learning and Instruction*, vol. 12, p. 529-556. 2002.

[Robert04] Robert, C. P., & Casella, G. *Monte Carlo statistical methods*. New York: Springer, vol. 319, 2004.

[Rosales11] Rosales-Morales, V.Y.; Alor-Hernández, G.; Juárez-Martínez, U. "An overview of multimedia support into JavaScript-based Frameworks for developing RIAs," *Electrical Communications and Computers (CONIELECOMP), 2011 21st International Conference on* , p. 66-70, Feb. 28 2011-March 2 2011.

[Shannon87] Robert E. Shannon, "Models and artificial intelligence", *Proceedings of the 19th conference on Winter simulation*, p. 16-24, December 14-16, 1987, Atlanta, Georgia, United States.

[SICS00] "Simulation Interoperability Standards Committee (SISC) of the IEEE Computer Society [C]". *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)* - IEEE Std 1516/1516.1-2000. Institute of Electrical and Electronics Engineers, Inc. USA. ISBN 0-7381-2619-5 SH94882, 2000/0-7381-2621-7 SH94883, 2000.

[Silberchatz91] Henry F. Korth, Abraham Silberchatz: *Database System Concepts*, cap. 12, p. 395-423. Ed. McGraw-HILL. Segunda Edición. 1991.

[Sinclair86] J. B. Sinclair, S. Madala, "A graphical interface for specification of extended queueing network models", *Proceedings of 1986 ACM Fall joint computer conference*, p. 709-718, November 1986, Dallas, Texas, United States.

[Sirota00] Sirota R. : "El oficio de alumno y la sociología", en *Políticas Instituciones y actores en educación*. Buenos Aires, Coedición Novedades Educativas, Centro de Estudios Multidisciplinarios. 2000.

[Stabley90] Don H. Stabley. *Assembler language for application programming.* Editor TAB Professional and Reference Books, 1990. ISBN 0830602828.

[Ståhl00] Ingolf Ståhl. "How should we teach simulation?" *Proceedings of the 2000 Winter*

Simulation Conference. Orlando, Florida, USA. 2000, p. 1602-1612.

[Ståhl01] Ingolf Ståhl. 2001. "GPSS: 40 years of development". In *Proceedings of the 33nd conference on Winter simulation (WSC '01).* IEEE Computer Society, Washington, DC, USA, p. 577-585.

[Steudel87] Harold J. Steudel, Taeho Park, "STAR*CELL: a flexible manufacturing cell simulator", *Proceedings of the 19th conference on Winter simulation*, p. 230-234, December 14-16, 1987, Atlanta, Georgia, United States.

[Ueno80] D. H. Uneo and W. Vaessen, "PASSIM: A Discrete-Event Simulation Package for Pascal", *Simulation*, vol. 36, no. 6. 1980.

[Vaughan04] Vaughan-Nichols, S. J. "Achieving wireless broadband with WiMax". *Computer*, vol.37, no. 6, pp. 10-13, June 2004.

[Vlatka00] Vlatka Hlupic. "Simulation software: an Operational Research Society survey of academic and industrial users". In *Proceedings of the 32nd conference on Winter simulation (WSC '00).* Society for Computer Simulation International, San Diego, CA, USA, p. 1676-1683. 2000.

[Watson12] Robert B. Watson. 2012. "Development and application of a heuristic to assess trends in API documentation". In *Proceedings of the 30th ACM international conference on Design of communication (SIGDOC '12).* ACM, New York, NY, USA, p. 295-302.

[White86] David A. White, "PCModel and PCModel/GAF—screen oriented modeling", *Proceedings of the 18th conference on Winter simulation*, p. 164-167, December 08-10, 1986, Washington, D.C., United States .

[Wiki13] Run-time Infraestructure (Simulation). Fuente: http://en.wikipedia.org/wiki/Run-Time_Infrastructure_(simulation). Última revisión: mayo 2013.

[Wolverine92] Wolverine Software Corporation. 1992. Using Proof Animation. Online. http://www.wolverinesoftware.com/ProofCreation.htm

[Wood99] Wood H. and Wood D. "Help seeking, learning and contingent tutoring". In *Computers and Education*, vol. 33, p. 153-163. 1999.

[Zeigler84] Bernard P. Zeigler, *Theory of Modelling and Simulation*, Krieger Publishing Co., Inc., Melbourne, FL, 1984

[Zeigler00] Zeigler, B. P., Praehofer, H., & Kim, T. G. (2000). *Theory of modeling and simulation: Integrating discrete event and continuous complex dynamic systems*. Academic Press.

[Zikic92] A. Zikic, B. Radenkovic, "An application of GPSS/FON in Teaching Simulation", *International Journal of Engineering Education*, vol. 8, no. 5, p. 355-366, 1992.

[Zikic96] Zikic, A. M.; Radenkovic, L. J.: "New Approach to Teaching Discrete Event System Simulation". *International Journal of Engineering Education*; vol 12, no. 6, p. 457-466. 1996.

Anexo 1

Sistema de clasificación automática de frutas

Uno de los procesos intermedios de una planta exportadora de frutas consiste en clasificar por tamaño las frutas para acondicionarlas con un embalaje que varía precisamente según el tamaño. La fruta es descargada en cintas transportadoras y en las mismas reciben un lavado a través de un circuito de caños ubicados en la parte superior de la cinta.

En las cintas, la fruta se dispersa por lo que el tiempo de arribo al punto de supervisión es aleatorio. Los tiempos de arribo pueden ser de 1, 2, 3, 4, 5 o 6 segundos con igual probabilidad. La acción de clasificación por tamaño la realiza un ojo robótico que tarda un tiempo aleatorio: 1 (25%) o 2 (25%) o 3(25%) o 4(25%) segundos con igual probabilidad.

Con esta información, se requiere construir una tabla de simulación, suponiendo el arribo de 10 frutas. La tabla debe desarrollar los tiempos entre arribos, el tiempo de arribo al ojo, el número de cada fruta, el momento de inicio de la clasificación, la duración de la clasificación y el momento de finalización de la clasificación.

Una vez completada la tabla, calcular:

- Tiempo total de la simulación.
- Tiempo de ocupación del ojo robótico.
- Tiempo ocioso del ojo robótico.
- Tiempo promedio de atención de las frutas.
- Tiempo máximo y mínimo de atención de las frutas.
- Cantidad máxima, mínimo y promedio de la frutas en la cola de espera.