Universidad Nacional de La Plata

Facultad de Informática

1999

Control de acceso y cambios en ambientes de aprendizaje de objetos.

Caso de estudio: LearningWorks

Trabajo de Grado de la Carrera de Licenciatura en Informática

Alumno: Alejandro Fernández

Director: Doctor Gustavo Rossi, LIFIA, UNLP.

Co-Directores: Doctora Adele Goldberg y David Leibs, Neometron inc.

Tabla de Contenidos

Capítulo 1

Introducción

Durante la última década, la tecnología de orientación a objetos a revolucionado todos los ámbitos de la ingeniería de software. Inicialmente adoptada solo en ámbitos académicos y de investigación se extendió para internarse fuertemente en la industria. En la actualidad toda herramienta o proceso de desarrollo tiene en cuenta los principios de la tecnología de objetos.

A pesar de la convicción con que gran número de grupos de desarrollo migran a esta tecnología, los resultados qué prometen tardan en alcanzarse. Es común escuchar de compañías que fracasan en sus intentos de incorporar la tecnología de objetos en sus procesos de desarrollo de software. Quienes dedican la mayor parte de sus tareas a develar el por que de este problema están convencidos que se trata de una falla en la formación.

La tecnología de objetos propone un cambios profundo en la realización del software. No se trata de un nuevo lenguaje o herramienta de programación sino de un nuevo proceso de concepción de aplicaciones. Para una integración efectiva de la tecnología de objetos en un grupo de construcción de software se debe comenzar desde el principio: aprender y entender el modelo. No basta con ser capaz de programar C++, Java o Smalltalk; hay que entender y estar convencido de los principios del modelo que da soporte a la tecnología de objetos. Esto conlleva a necesidad de fortalecer y mejorar las técnicas de aprendizaje potenciándolas para las nuevas características de este paradigma.

La búsqueda de nuevos enfoques para el aprendizaje de tecnología de objetos ha llevado a la formación de diversos grupos de investigación y a la aparición de conferencias y publicaciones científicas al respecto. Algunos de ellos centran sus esfuerzos en los aspectos pedagógicos, como es el caso del Proyecto Pedagogical Patterns, otros en el desarrollo de herramientas de aprendizaje, como el caso de la línea sobre orientación a objetos de la conferencias ITiCSE, otros en el aprendizaje de objetos en general, como es el caso de el Educator's simposium en OOPSLA.

Este trabajo expone los resultados de nuestros esfuerzos en pos de mejorar el proceso de aprendizaje de tecnología de objetos. El contexto de trabajo son las herramientas de software. En particular hemos adoptado LearningWorks, una herramienta ideada y desarrollada por la Doctora Adele Goldberg y su empresa Neometron y originalmente financiada por Mitsubishi Electronics. LearningWorks es un ambiente especialmente desarrollado para explorar y aprender acerca del proceso de desarrollo de software en objetos.

Las extensiones presentadas aquí, discutidas y pautadas en acuerdo con los autores y propietarios de la herramienta, apuntan a fortalecer el ambiente y sus mecanismos de exploración del paradigma de objetos.

1.1 Organización

Este documento se organiza en 11 capítulos. El capítulo 2, *"Aprendizaje de Tecnología de Orientación a Objetos"* expone los aspectos mas relevantes de aprendizaje de tecnología de objetos, estableciendo las bases para el desarrollo de toda herramienta de aprendizaje. El Capítulo 3, *"Programación Orientada a Objetos"*, describe las características principales de los lenguajes de programación en objetos haciendo énfasis en su uso para el aprendizaje. El Capítulo 4, *"LearningWorks: Un Ambiente de aprendizaje de Objetos"* presenta LearningWorks, un ambiente para la exploración y programación de aplicaciones de objetos. El Capítulo 5, *"El Framework de LearningWorks"*, detallas algunas de las características de implementación del ambiente de forma de sentar las bases de discusión de los capítulos siguientes. El Capítulo 6, *"Modelo de Objetos de Smalltalk 80"*, describe los principios fundamentales de la construcción del modelo de objetos que da soporte a LearningWorks, Smalltalk 80. Los elementos aportados por este capítulo son esenciales para entender las limitaciones de LearningWorks expuestas en el Capítulo 7, *"Limitaciones de LearningWorks"*. El Capítulo 8, *"Integridad y Visión: un Nuevo Modelo"*, introduce un nuevo modelo que soporta los cambios necesarios para salvar las limitaciones expuestas con anterioridad. El Capítulo 9, *"Nuevo Framework de LearningWorks"*, detalla los cambios en el diseño del framework de LearningWorks que debieron efectuarse en pos de implementar el modelo descripto en el Capítulo 8, . Finalmente el capítulo 10 presenta conclusiones y el Capítulo 11 enumera las fuentes bibliográficas cubiertas. Cada capítulo cuenta en su final con un listado de las fuentes bibliográficas mas relevantes del mismo.

Capítulo 2

Aprendizaje de Tecnología de Orientación a Objetos

La tecnología de Orientación a Objetos se ha convertido en un tema curricular crítico para la mayoría de las carreras en el área de la ingeniería de software. Este tópico ha superado las barreras de lo académico para convertirse en una necesidad diaria y palpable. Ello se debe en gran parte a la cantidad de herramientas y ambientes de software que han adoptado esta tecnología como base. Este cambio que afecta, entre otros, a lenguajes, ambientes, bases de datos y metodologías de diseño se traslada directamente a la industria en forma de una necesidad imperiosa de formación. El paradigma de objetos posibilita también la aparición de tecnologías satélites como la programación por componentes, la integración de modelos de objetos con modelos relacionales y los objetos distribuidos, haciendo crítica la necesidad de formación.

El paradigma de orientación a objetos afecta todas las áreas del desarrollo de software. Inicialmente lanzado como una técnica de programación, se ha vuelto el contexto fundamental para la concepción de todo proyecto de software abarcando desde el análisis hasta el mantenimiento y prueba. Su alcance que originalmente se veía limitado por la capacidades de los lenguajes de programación disponibles comprende ahora todas las áreas. En la actualidad se dispone de lenguajes para el desarrollo de aplicaciones de tiempo real, distribuidas, concurrentes, con acceso a base de datos, con soporte gráfico, de bajo nivel, para contexto comerciales, para comercio electrónico y para programación distribuida en Internet.

Son muchas las universidades que en la actualidad incluyen en su currícula tópicos de orientación a objetos. Las que no, se encuentran en proceso de adaptar sus currículas de manera acorde. Un fenómeno similar se percibe en la industria y se caracteriza por una fuerte predisposición a la migración a nuevos lenguajes y herramientas que soporten tecnología de objetos. Este fenómeno se ve aumentado por la constante evolución de los lenguajes y herramientas de desarrollo que día a día incorporan más características de dicha tecnología forzando el cambio. Características incorporadas

originalmente como extensiones de uso opcional ahora se vuelven principios fundamentales de lenguajes y herramientas.

2.1 Integración en la Currícula

Hay un debate interesante respecto al momento adecuado para la introducción de la tecnología de orientación de objetos. Si se trata de un contexto industrial entonces la única respuesta posible es "ahora". Las necesidades de dicho contexto no coinciden generalmente con las del aprendizaje, haciendo más duro cualquier movimiento en esta dirección. Sin embargo, en el ámbito académico, los tiempos y mecanismos para la integración de tecnología de orientación a objetos se encuentran en constante evolución. Muchas carreras están moviendo sus cursos introductorios de objetos a los primeros años. Cursos que originalmente tenían como objetivo principal el aprender programación estructurada (CS1) o estructuras de datos (CS2) basándose en lenguajes como C y Pascal ahora enfocan programación orientada a objetos en C++, Java o Smalltalk. Si bien este cambio se presenta normalmente como una migración de lenguaje, es de hecho mucho más profundo. La adopción de tecnología de orientación a objetos en cursos introductorios va mas allá del aprendizaje de nuevas construcciones de programación e involucra un profundo cambio en la forma en que el software es desarrollado. El objetivo del movimiento es la concepción de un modelo de resolución de problemas donde las herramientas fundamentales son *"los objetos"*. En este nuevo contexto, la programación en sí pierde su papel central para dejar lugar al diseño de modelos de objetos. El lenguaje y sus construcciones pierden importancia para dar lugar a herramientas de construcción y simulación más simples pero no por eso menos poderosas entre las que podemos mencionar *"juegos de roles"* y *"exploración de micro-mundos"*.

El desarrollo de software utilizando objetos se introducía en sus orígenes como un paso más en la evolución de las técnicas estructuradas al que se llegaba mediante construcción y utilización de Tipos Abstractos de Datos (ADT). Esto derivaba en cursos híbridos donde se combinaban técnicas de diseño y programación estructuradas con solo algunas de las características particulares de objetos (a lo sumo encapsulamiento y herencia). Se ha demostrado en innumerables trabajos de investigación y conferencias de educación que este enfoque es deficiente y en algunos casos contraproducente. Algunos aspectos de la programación estructurada como por ejemplo la separación de datos y control o la unicidad de nombres de funciones y procedimientos son contrarios a los principios de la programación orientada a objetos. Para estos ejemplos como otros, el conocimiento de principios estructurado no solo carece de utilidad al momento de aprender a trabajar con objetos sino que, muy

por el contrario, se vuelve una traba. Es debido a esto que el camino mas seguido y recomendado por expertos en el aprendizaje de esta nueva tecnología es *"objetos primero"*. Experiencias llevadas a cabo, demuestran que estudiantes sin previo conocimiento de programación pueden rápidamente entender y aplicar los principios de la construcción de software en objetos. Esto se debe a la simplicidad de sus conceptos básicos: objeto, conocimiento, responsabilidad, colaboración. Este es también el enfoque adoptado en la actualidad por los mas distinguidos autores en el área.

El éxito en el aprendizaje de tecnología de objetos depende principalmente de la combinación de estos dos factores: elección del momento adecuado y enfoque. Sin duda la combinación que demuestra mejores resultados se basa en la introducción temprana del paradigma con un enfoque *"objetos primero"* haciendo énfasis en el diseño. Un contexto adecuado combina este enfoque con técnicas de exploración y lenguajes de programación adecuados que respeten y enriquezcan los principios del paradigma. La riqueza en ejemplos teniendo en cuenta tanto su variedad como su calidad es fundamental. En el aprendizaje exploratorio, un ambiente rico en ejemplos provee una fuente importante de conocimiento. En el caso del diseño y la programación con objetos, la riqueza de los modelos utilizados como ejemplos se transmite progresivamente al alumno generando en el mismo *"gusto"* por el buen diseño. La presentación de heurísticas y patrones de diseño es de gran utilidad para proveer a los alumnos de herramientas de evaluación que suplanten la experiencia de diseño que aún no han adquirido.

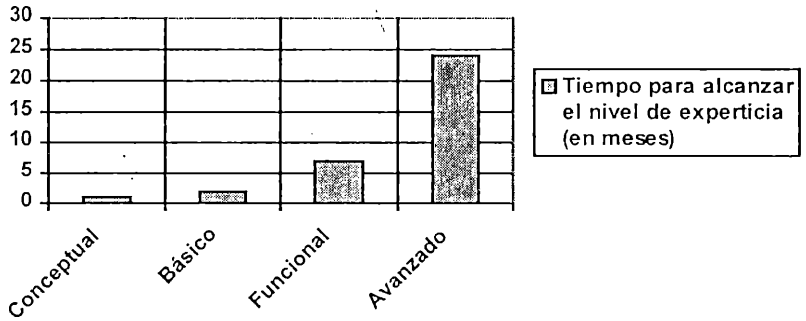


Ilustración 2.1 - **Fuente: Succeeding with objects - Goldberg, Rubin**

Finalmente es de fundamental importancia fomentar el desarrollo de actividades de trabajo cooperativo. Por ser el diseño una actividad creativa, se ve ampliamente enriquecida por la discusión, evaluación y comparación de alternativas. Las actividades de cooperación exponen a los alumnos a condiciones de trabajo reales del ámbito de la ingeniería y a su vez actúan como agentes motivadores. Las discusiones de diseño y la evaluación cooperativa del producto de sus pares enriquece a todos los

participantes. Un entorno de trabajo cooperativo permite que alumnos más avanzados actúen como mentores de quienes encuentran mayores dificultades, poniendo a prueba y afirmando a su vez sus propios conocimientos.

Estudios estadísticos muestran que para alcanzar nivel funcional en orientación a objetos se requieren entre seis y ocho meses de estudio mientras que el tiempo requerido para alcanzar nivel avanzado depende fuertemente de cuanto se involucre el estudiante en proyecto reales de desarrollo (Ilustración 2.1).

2.2 Principios Fundamentales

En lo que respecta a alcanzar nivel funcional, cinco son los pilares de la tecnología de objetos: objeto, composición, herencia, delegación y polimorfismo. Los mismos conforma la base de todo curso introductorio.

Objetos: Los objetos (generalmente modelados en clases) son las construcciones principales del paradigma. Es fundamental enfatizar en la integración de comportamiento (operaciones) y conocimiento (datos) que se da en los mismos. El encapsulamiento de operaciones y datos y el ocultamiento de su implementación son dos características de la cuales los objetos consiguen su poder. Es esencial que los alumnos se familiaricen con ellas y entiendan su impacto en la construcción de software extensible, modificable y reusable. Esto puede conseguirse mediante la ejercitación en la construcción con objetos preexistente de los que se desconoce su implementación. Los micro-mundos orientados a objetos representan un excelente contexto donde llevar a cabo dicha actividad.

Composición y Herencia: Los objetos, por ser unidades de construcción altamente cohesivas, se convierten en importantes candidatos para la construcción de software reusable. La construcción en base a la utilización de componentes preexistentes es una de las actividades mas interesantes y productivas del paradigma. Dos mecanismos proveen la base para este mecanismo: la herencia y la composición. Los alumnos deben ser capaces de detectar oportunidades para la aplicación de estos mecanismos teniendo claro conocimiento de sus similitudes, diferencias y de las fuerzas involucradas.

Delegación y polimorfismo: El aspecto de diseño mas simple introducido por la tecnología de objetos es la delegación. Combinada con la posibilidad de que objetos de distintas clases implementen operaciones a las que se hace referencia por un mismo nombre (polimorfismo) permite la construcción de aplicaciones extensibles y modificables. La construcción de objetos simples y reusables se basa principalmente en la capacidad del diseñador de asignar responsabilidades a los objetos de forma que

cada uno implemente un conjunto de operaciones coherente y limitado. La riqueza en el diseño se consigue entonces por la combinación de objetos que cooperan delegando operaciones unos en otros. Cuando las responsabilidades de los objetos se determinan de forma correcta, se consiguen diseños donde las componentes (los objetos) pueden evolucionar de forma independiente. El diseño de objetos basado en responsabilidades, introducido por Wirfs-Brock, saca máximo provecho de estas características. Por la simplicidad de las heurísticas que propone, el diseño basado en responsabilidades es el que mejor se adecua a un ambiente de aprendizaje.

2.3 Referencias

[1] Beck, Kent, Cunningham, Ward. A Laboratory for Teaching Object-Oriented Thinking. Proceedings de OOPSLA'89. ACM Press, 1989.

[2] Cross, James H., Phillips, Thomas. Succesfully Integrating Traditional and Object-Oriented Approaches with Ada 95. Proceedings de SIGCSE'96 (19-23). ACM PRESS, 1996. ISSN 0097-8418

[3] Culwind, Fintan. Object Imperatives!. Proceedings de SIGCSE'99 (31-36). ACM PRESS, 1999. ISSN 0097-8418

[4] Goldberg, Adele; Rubin, Kenneth S. Succeeding with Objects. Addison Wesley, 1985. ISBN 0-201-62878-3

[5] Parlante, Nick. Teaching with Object Oriented Libraries. Proceedings de SIGCSE'97 (140-144). ACM PRESS, 1997. ISSN 0097-8418

[6] Proulx, Viera. Traffic Simulation: A Case Study for Teaching Object Oriented Design. Proceedings de SIGCSE'98 (48-52). ACM PRESS, 1998. ISSN 0097-8418

[7] Reek, Keneth. Teaching Inheritance versus Inclusion to First Year Computer Science Students. Proceedings de SIGCSE'96 (24-26). ACM PRESS, 1996. ISSN 0097-8418

[8] Resnik, Mitchel. Turtles, Termites, and Traffic Jams. Explorations in Massively Parallel Microworlds. MIT Press, 1994. ISBN 0-262-18162-2

[9] Riel, Arthur J. Object-Oriented Design Heuristics. Addison Wesley, 1996. ISBN 0-201-63385-X

[10] Rumbaugh, James, Blaha, Michael, Lorensen, Wiliam, Eddy, Frederick, Premerlani, William. Object-Oriented Modeling and Design. Prentice Hall Engineering, Science & Math, 1990. ISBN 0-13-629841-9

[11] Wirfs-Brock, Rebecca, Wilkerson, Brian, Wiener, Lauren. Designing Object-Oriented Software. Prentice Hall Professional Technical Reference, 1990. ISBN 0-13-629825-7.

[12] Woodman, Mark, Davies, Gordon, Holland, Simon. The Joy of Sotware - Starting with Objects. Proceedings de SIGCSE'96 (88-92). ACM PRESS, 1996. ISSN 0097-8418

Capítulo 3

Programación Orientada a Objetos

Durante los últimos 10 años se ha llevado a cabo un cambio mayor en el diseño de los lenguajes de programación, dejando los lenguajes procedurales (por ejemplo Pascal o C) para concentrarse en lenguajes orientados a objetos (como C++, Eifel, Smalltalk, Java). El objetivo principal de este cambio, es capturar las potenciales ventajas de la tecnología de orientación a objetos entre las que se encuentra un aumento en la productividad de los programadores, un aumento en la robustez de las aplicaciones, una forma más simple y rápida de modelar los objetos del mundo real e importantes niveles de reuso de código.

Concurrentemente al desarrollo de la tecnología de objetos, se ve un importante avance en el desarrollo de los ambientes de programación. Mientras que hasta ahora estos sistemas simplemente contaban con un editor de texto y un compilador, los ambientes de programación modernos proveen además facilidades como manejo de código fuente, manejo de librerías, soporte para el trabajo en grupo, control de versiones y herramientas que integran edición, compilación, prueba, depuración.

Al tiempo que los lenguajes de programación en objetos evolucionan, mucho esfuerzo se dedica a intentar integrarlos con ambientes avanzados. En general el enfoque ha sido adaptar ambientes de software existentes a lenguajes orientados a objetos. Sin embargo, esos intentos no han podido capturar las ventajas que ofrece la orientación a objetos. La razón más significativa de este fracaso es que los sistemas existentes se concentran en abstracciones que solo son apropiadas para lenguajes procedurales. Consecuentemente, proveen soporte para el desarrollo de programas procedurales, manejo de archivos, organización de la información de prueba, etc. Un ambiente de programación orientado a objetos debe proveer soporte para clases y objetos como abstracciones fundamentales. Intentar utilizar mecanismos desarrollados para lenguajes procedurales no es necesariamente apropiado.

Un simple ejemplo se encuentra en la etapa de prueba del software. Un ambiente de programación procedural proveerá soporte para la prueba de programas procedurales. Esto incluirá: preparar datos de entrada, capturar la salida y compararla con la salida esperada. Un ambiente de programación en objetos proveerá un mecanismo de prueba de objetos. Esto permitirá la invocación interactiva de las operaciones de interface del objeto, algo muy diferente a lo que se requiere para ambientes procedurales. Otra diferencia potencial se encuentra en el hecho de que un ambiente de programación en objetos proveerá soporte para desarrollo incremental y descartará la necesidad de escribir programas de prueba.

La falta de ambientes realmente orientados a objetos ha traído grandes dificultades para el aprendizaje de esta tecnología. En particular, los alumnos encuentran grandes dificultades conceptuales y tienden a escribir programas procedurales en lenguajes orientados a objetos. Esto prevalece particularmente cuando se trata de lenguajes como C++ que son aún compatibles con sus versiones anteriores no orientadas a objetos. Si se espera que los alumnos se integren por completo a la tecnología de orientación a objetos, entonces, se les debe proveer de lenguajes y ambientes apropiados.

3.1 Evolución de los Lenguajes Orientados a Objetos

Durante los últimos años se ha visto la casi universal adopción del paradigma de programación orientada a objetos. Aunque este enfoque es usado desde hace largo tiempo (desde que Simula fue desarrollado en 1960) pasó mucho hasta que se reconoció su importancia en el desarrollo de software. Por un largo período Smalltalk fue el único lenguaje de programación orientada a objetos ampliamente disponible. El mismo era visto generalmente como un lenguaje aplicable solo a la investigación y ciertamente inapropiado para la enseñanza. Las razones principales para quienes mantenían esta postura eran su baja performance y sintaxis poco común en comparación con otros lenguajes. Durante la década de los 80, el desarrollo de otros lenguajes así como los avances conseguidos en el área de los compiladores resultaron en un aumento en la investigación y discusión acerca de los conceptos de orientación a objetos.

La programación orientada a objetos es vista por muchos como la mejor respuesta a la creciente complejidad del software. Muchos de los nuevos lenguajes, que han evolucionado en la última década incluyen conceptos de orientación a objetos. Algunos de ellos solo extienden lenguajes existentes como C o Pascal incorporando soporte para objetos. Otros lenguajes como Eiffel, Sather o Java han sido desarrollados teniendo como principal meta la orientación a objetos. C++, el sucesor orientado a objetos de C, se convirtió rápidamente en el más popular para proyectos industriales,

reemplazando a C. La única razón para la popularidad de C++ es la inclusión de conceptos de orientación a objetos.

Muchas universidades han adoptado un lenguaje orientado a objetos para sus cursos introductorios, otras están considerando seriamente hacerlo. Sin embargo existen reportes de muchas instituciones acerca de las dificultades que encuentran con este enfoque. El más común tiene que ver con la falta de ambientes de desarrollo apropiados. Fuera de la discusión acerca de las características de algunos lenguajes de programación en particular, no parece haber interés alguno en mejorar las características del ambiente de programación. Sin embargo este tiene mayor impacto en el aprendizaje que el lenguaje mismo.

3.2 Ambientes de Programación Orientada a Objetos

En sistemas utilizados en muchas universidades en la actualidad la programación está basada en un ambiente de línea de comando o textual donde el soporte para las distintas tareas del proceso de programación es provisto por distintas herramientas. Estas herramientas (típicamente un editor, un compilador, un depurador y algunos scripts para simplificar el proceso), se basan en conceptos desarrollados en los 60s y no han cambiado desde su introducción. Esta situación ha sido en parte mejorada por la aparición de ambientes gráficos que integran gran parte de las herramientas de programación y que se encuentran disponibles principalmente para computadoras personales. Dichos ambientes permiten reducir dramáticamente la carga de organización de los procesos de programación al integrar editores, compiladores y depuradores para formar sistemas coherentes. Todas las herramientas funcionan coordinadas potenciando el proceso de desarrollo.

Para mejorar el proceso de desarrollo de software en objetos, es necesario integrar los lenguajes con ambientes de programación avanzados. Si bien Smalltalk desde sus principios integra ambiente y lenguaje para conformar una herramienta muy sofisticada, otros lenguajes desarrollados mas recientemente parecen haber pasado por alto este aspecto apuntando principalmente a una interface de texto. El avance impetuoso de la tecnología en los últimos tiempos a dejado al descubierto esta falencia. Esto se ha convertido en una importante motivación para el desarrollo de ambientes gráficos e integrados para lenguajes orientados a objetos. Son varios los ambientes que en la actualidad dan soporte integrado para la programación en objetos en C++, Eiffel, Oberon y Java. Sin embargo, a pesar de este esfuerzo, los ambientes siguen siendo deficientes.

El problema principal reside en no haber entendido ni utilizado las características y requerimientos particulares de los lenguajes orientados a objetos. Las principales deficiencias de los

ambientes de programación existentes son la falta de soporte en el manejo de objetos y la insuficiencia en los mecanismos para la manipulación y visualización. Además, su utilización para el aprendizaje se ve obstaculizada por la falta de mecanismos de configuración a condiciones particulares y por la deficiencia con la que implementan los principios del paradigma.

3.2.1 Soporte para Trabajar con Objetos

Los ambientes de programación tradicionales facilitaban el diseño y la construcción de programas procedurales. La entidad básica era el código fuente y la funcionalidad de los ambientes giraba en torno a la manipulación del mismo. La meta en ese contexto era producir un programa, la descripción de un algoritmo con exactamente un punto de entrada y que solo podía ser construido luego de que todas sus partes estuvieran completas. No había ninguna forma de manejar objetos activos dentro del ambiente de desarrollo ya que los mismos no pueden existir independientemente de la ejecución activa de un programa. Toda la información disponible fuera del contexto de ejecución se encontraba en forma de archivos de datos. Consecuentemente, todo lo que el ambiente hacía era tratar con archivos.

Cuando estos ambientes fueron adaptados a trabajar con lenguajes orientados a objetos los archivos de fuentes fueron reemplazados por definiciones de clases. Típicamente, se necesitan mas archivos en un ambiente orientado a objetos que en uno procedural; además, hay más relaciones entre estos archivos (por ejemplo herencia, composición, etc). En consecuencia debieron, agregarse a los ambientes herramientas para manejar archivos de clases y algunas de sus relaciones. Sin embargo, el paradigma general de los ambientes no fue cambiado, el ambiente siguió siendo utilizado para construir una aplicación que tiene un solo punto de entrada y que solo puede ser construida después de que todas sus partes están completas. Siguió sin adoptarse el paradigma de orientación a objetos para los ambientes en sí.

El paradigma de orientación a objetos se basa en la idea de que los objetos existen independientemente y que sobre ellos se pueden invocar operaciones. Consecuentemente, un usuario de un ambiente verdaderamente orientado a objetos debe ser capaz de crear objetos de alguna clase disponible y de forma interactiva manipularlo, invocar alguna de sus operaciones, explorar su composición, etc. Debe también ser posible componer objetos a partir de objetos ya creados. Esto conduce a la posibilidad de tener desarrollo incremental de aplicaciones, cualquier clase individual puede ser probada independientemente tan pronto como este lista. De esta manera, el proceso de prueba se vuelve mucho mas flexible que en los ambientes procedurales. En la mayoría de los

ambientes existentes, los objetos deben ser embebidos en programas de características procedurales que actúan como contexto de prueba y que invocan sus operaciones.

En resumen, un ambiente para el desarrollo con objetos debe:

- hacer de las clases su principal mecanismo para la estructuración de código fuente;
- hacer de los objetos entidades sobre las que se puede operar.

Hasta el momento solo Smalltalk y algunos ambientes para Java proveen este tipo de funcionalidad.

3.2.2 Soporte de Visualización

Un problema presente en de la mayoría de los ambientes existentes es la falta de mecanismos para la visualización de objetos. Para mostrar las relaciones entre los objetos y las clases deberían usarse técnicas gráficas. Por ejemplo, podría mostrarse las relaciones de herencia y composición utilizando diagramas UML o similares aumentando la expresividad del ambiente. Aunque las relaciones entre objetos son el factor esencial en el desarrollo orientado a objetos, es muy poco el soporte que los ambientes brindan para su visualización y manipulación.

Debe ser posible utilizar de manera arbitraria representaciones textuales o gráficas de las clases. Eso implica, por ejemplo, que debe ser posible editar una relación de herencia gráficamente y que automáticamente se actualice la representación textual de las clases. Lo mismo debería valer para el caso opuesto, los cambios en las representaciones textuales deberían reflejarse automáticamente en la representación gráfica.

Los ambientes de programación hoy en día no tienen esta funcionalidad. Los sistemas gráficos de muchos de los lenguajes soportan solo débilmente este tipo de funcionalidad. La mayoría de ellos proveen una buena integración de las herramientas pero carecen de soporte para la manipulación interactiva y visual de objetos. Los ambientes de desarrollo mas avanzados como Visual C++ y Delphi, proveen soporte gráfico pero solo para la construcción de componentes de interface de usuario delegando el modelado de objetos a un segundo lugar. En estos ambientes profesionales ni siquiera es posible manipular directamente instancias; VisualAge de IBM es uno de los ambientes mas avanzado en este sentido.

El único ambiente que provee toda esta funcionalidad es Smalltalk. Se puede crear objetos y usarlos de forma interactiva. El ambiente Portia para Smalltalk provee mecanismos muy avanzados para la manipulación directa de instancias.

3.2.3 Adaptabilidad a las Condiciones de Aprendizaje

La mayor desventaja que presentan los ambientes de programación en objetos disponibles al momento respecto a su utilización académica es la imposibilidad de poder adaptarlos a condiciones específicas de aprendizaje. La mayoría de ellos son herramientas de carácter profesional e incluyen tanta funcionalidad que se convierten en un obstáculo para el aprendizaje. La riqueza en librerías de clases y la flexibilidad de sus herramientas vuelve a estos ambientes demasiado complicados como para ser utilizados por los alumnos, quienes encuentran más dificultades para aprender a utilizar el ambiente que para aprender los conceptos de la tecnología subyacente.

Por ser herramientas de uso profesional muchas de ellas quedan fuera del alcance académico. La posibilidad de integrarlas en un contexto de aprendizaje donde los recursos económicos son limitados, depende de la disponibilidad de versiones no comerciales o de planes de cooperación entre las empresas productoras y las universidades.

Un buen ambiente de aprendizaje permite que el docente lo configure para asegurar que el alumno centra su atención en los puntos de interés y que obtiene el mayor provecho posible de las herramientas disponibles. Todo aquello que no es esencial para el aprendizaje debe ser evitado y escondido. En situaciones de aprendizaje exploratorio, esto se vuelve sumamente importante. Los micro-mundos han demostrado ser muy poderosos en este aspecto; Logo y Karel son dos grandes exponentes. La experiencia obtenida después de años de utilizarlos en las mas variadas circunstancias los convierte en modelos a imitar.

La implementación de micro-mundos orientados a objetos requiere que el ambiente de programación pueda adaptarse para trabajar con objetos de un dominio particular y permitir actividades de exploración interactiva con representaciones gráficas. A su vez, dado que como parte del trabajo en micro-mundo los alumnos modifican y extienden el mismo es fundamental que el ambiente pueda configurarse para asegurar su consistencia aún cuando el alumno lleve a cabo acciones inesperadas.

Los lenguajes y ambientes de programación existentes no permiten esta clase de configuración. Solo herramientas desarrolladas especialmente (como los mencionados Logo y Karel)

sirven para la exploración de micro-mundos. Sin embargo estos se vuelven inútiles una vez que los alumnos han alcanzado cierto nivel de experticia haciendo necesario el uso de alguna otra herramienta. Esto agrega carga innecesaria al proceso de aprendizaje.

Para que un ambiente evolucione junto con los alumnos debe proveer formas simples de adaptarlo a las necesidades particulares del los mismos. Para que esto no sea tan complicado como desarrollar un lenguaje y ambiente desde cero, se debe contar con mecanismos que permitan adaptar el ambiente y lenguaje de forma incremental y sin recurrir a herramientas mas complejas. Los mecanismos de reflexión, solo implementados por algunos pocos lenguajes, son la alternativa mas atractiva. Un lenguaje con posibilidad de reflexión cuenta con un modelo extensible y modificable de si mismo. Lo más interesante de este tipo de ambientes es que ellos son la única herramienta necesaria para aprovechar esta capacidad. El propio ambiente se utiliza para conseguir una versión mejorada de si mismo. Para quienes cuentan con experiencia de desarrollo utilizando el ambiente, modificarlo es solo tan complicado como entender la parte su modelo a la que el cambio afecta. A fin de cuentas el ambiente es solo una aplicación con las mismas características que aquellas obtenidas a partir de él. Un ambiente totalmente reflexivo no presenta limitaciones en lo que pueda conseguirse de él. Lamentablemente solo Smalltalk posee estas características. Otros lenguajes (por ejemplo Java) evolucionan lentamente en esta dirección.

3.2.4 Pureza de Implementación del Paradigma

El paradigma de orientación a objetos se basa principalmente en los conceptos de *objeto* y de colaboración por *pasaje de mensajes*. Estos mismos se utilizan a lo largo de todas las etapas del proceso de desarrollo. De su alto nivel de abstracción proviene el poder del paradigma, por esta razón, es fundamental que los lenguajes implementen estos mecanismos lo mas puramente posible.

Si bien, estas construcciones pueden simularse con construcciones de lenguajes estructurados (registros e invocación de procedimientos) los resultados no son los mismos. El programador se ve obligado a moverse constantemente entre construcciones con distinto nivel de abstracción. Por un lado modela en objetos, y por otro, se ve en la necesidad de trasladar sus modelos a construcciones más primitivas. Mientras que el paradigma que utiliza para modelar le permite olvidarse de los mecanismos de manejo de memoria e invocación de operaciones, el paradigma de programación impone restricciones al respecto, limitando así los resultados. En el aprendizaje, esta falencia de los lenguajes es aún mas crítica ya que permite a los alumnos realizar operaciones fuera del paradigma de objetos. Sin duda, esto pone en riesgo el proceso de aprendizaje. Cuando los alumnos cuentan con una

formación previa en programación estructurada, la falta de pureza en el lenguaje se convierte en una amenaza a su formación en objetos.

Smalltalk es el lenguaje que implementa con mayor pureza el paradigma. El mismo da soporte a todas sus construcciones y no introduce ningún mecanismo que pueda atentar contra los principios de la orientación a objetos. Su manejo de memoria transparente (conseguido gracias al dispositivo de *"recolección de basura"*) libera al programador de esta carga. Dadas las características del lenguaje, los modelos de diseño pueden convertirse directamente en modelos de implementación, acelerando así el proceso de desarrollo.

Java es el lenguaje que sigue a Smalltalk en este aspecto. Su deficiencia mas importante esta dada por la diferenciación de los tipos básicos y las clases. El lenguaje da soporte a todas las construcciones del paradigma pero también incorpora conceptos de la programación estructurada. Las estructuras de control y los tipos primitivos se tratan de manera diferente al resto de las construcciones agregando una carga innecesaria de programación. A diferencia de Smalltalk, Java implementa chequeo estático de tipos, característica que suma puntos a su favor.

De los lenguajes de mayor popularidad, C++ es el que tiene mas construcciones fuera del paradigma. Por ser un sucesor de C, mantiene todas sus características haciendo posible la combinación de técnicas estructuradas con técnicas de orientación a objetos. La falta de mecanismos de alto nivel obligan al programador a preocuparle por alocación y liberación de memoria aumentando innecesariamente la complejidad de los programas.

3.3 Elección del Primer Lenguaje de Programación

Desde los principios de la informática se pone gran énfasis en la elección del primer lenguaje de programación. Esta elección es muy importante en la formación de los alumnos.

De lo expuesto en párrafos anteriores se deduce que Smalltalk es, de los existentes, el lenguaje más recomendable para el aprendizaje de tecnología de objetos. Muchas universidades lo han adoptado como su primer lenguaje de programación. En Argentina es sin duda el mas utilizado.

Aún cuando el potencial del ambiente permite que sea adaptado a condiciones particulares de aprendizaje, el costo de ello es muy elevado. La inversión de tiempo requerida para este tipo de configuraciones es muy alto. Esto ha originado durante los últimos años la aparición de nuevas versiones de Smalltalk de carácter académico que incorporan facilidades para soportar la exploración

de micro-mundos, simulaciones gráficas, realidades virtuales, etc. Los exponentes mas populares son LearningWorks y mas actualmente Squeak.

3.4 Referencias

[1] Arnow, David M., Weiss, Gerald. Introduction to Programming Using Java : An Object-Oriented Approach. Addison-Wesley Pub Co, 1998. ISBN: 0201311844

[2] Bergin, Joseph, Stehlik, Mark, Roberts, Jim, Pattis, Richard. Karel++: A Gentle Introduction to the Art of Object-Oriented Programming. John Wiley & Sons, 1996. ISBN: 0471138096.

[3] Eckel, Bruce. Thinking in C++. Prentice Hall Computer Books, 1995. ISBN: 0139177094.

[4] Eckel, Bruce. Thinking in Java. Prentice Hall Computer Books, 1997. ISBN: 0136597238.

[5] Flanagen, David. Java in a Nutshell : A Desktop Quick Reference (The Java Series). O'Reilly & Associates 2nd edition, 1997. ISBN: 156592262X

[6] Goldberg, Adele, Robson, David. Smalltalk-80. The Language and its Implementation. Addison Wesley, 1983. ISBN 0-201-11371-6

[7] Kölling, M. Koch, B y Rosenberg, J. Requirements for a First Year Object-Oriented Teaching Language. Proceedings de SIGCSE'95 (173-177). ACM PRESS, 1995. ISSN 0097-8418

[8] Kolling, Michael, Rosenberg, John. An Object-Oriented Program Development Environment for the First Programming Course. Proceedings de SIGCSE'96 (83-87). ACM PRESS, 1996. ISSN 0097-8418

[9] Lalonde, Wilf R, Pugh, John R. Inside Smalltalk: Volume 1. Prentice Hall Engineering, Science & Math, 1990. ISBN 0-13-468414-1

[10] Lalonde, Wilf R, Pugh, John R. Inside Smalltalk: Volume 2. Prentice Hall Engineering, Science & Math, ISBN 0-13-465964-3

[11] Lalonde, Wilf R, Pugh, John R. Smalltalk/V: Practice and Experience. Prentice Hall Engineering, Science & Math, 1993. ISBN 0-13-814039-1

[12] Paper, Seymour A. Sculley, John. Mindstorms : Children, Computers, and Powerful Ideas. Basic Books (Sd), 2nd edition 1999. ISBN: 0465046746

[13] Parlante, Nick. Teaching with Object Oriented Libraries. Proceedings de SIGCSE'97 (140-144). ACM PRESS, 1997. ISSN 0097-8418

[14] Resnik, Mitchel. Turtles, Termites, and Traffic Jams. Explorations in Massively Parallel Microworlds. MIT Press, 1994. ISBN 0-262-18162-2

[15] Savitch, Walter, Szvitch, Walter, Johnsonbaugh, Richard. Java: An Introduction to Computer Science and Programming. Prentice Hall, 1998. ISBN: 0132874261.

[16] Skublics, Suzanne, Klimas, Edward, Thomas, David. Smalltalk With Style. Prentice Hall, 1996. ISBN 0-13-165549-3

[17] Stroustrup, Bjarne. The C++ Programming Languaje. Adisson Wesley Publishing, 1992. ISBN: 0201889544

Capítulo 4

LearningWorks: Un Ambiente de Aprendizaje de Objetos

El proyecto LearningWorks empezó en 1994 como respuesta a la dificultad que la tecnología de objetos tenía en alcanzar sus metas. Los usuarios corporativos de la tecnología de objetos se frustraron por su inhabilidad de construir sistemas de software robustos y de fácil mantenimiento. En algún momento o en otro todos culparon de sus fallos a la falta de componentes probadas, imposibilidad de aplicar técnicas de prototipo, falta de know-how para hacer pruebas de aplicaciones hechas con lenguajes que soporten binding dinámico y la falta de programadores experimentados.

El proyecto atacó este problema como una gran oportunidad para el desarrollo de material educativo avanzado. Su objetivo era el desarrollo de un nuevo ambiente para la creación y distribución de material curricular, que pueda explotar las capacidades del trabajo en equipos. Este ambiente es LearningWorks

4.1 Motivación de LearningWorks

La premisa que condujo al desarrollo de LearningWorks es que uno puede aprender acerca de la construcción de sistemas antes de aprender acerca de programación (algoritmos, estructuras de datos). No se deben construir sistemas y especialmente no se debe enseñar construcción de sistemas sin un ambiente que integre la descripción de partes de software, su implementación, reuso e implementación. Construir sistemas sin un ambiente con esas características es lisa y llanamente muy complicado. Enseñar sin un ambiente como ese puede ser contraproducente.

LearningWorks está pensado para soportar tanto la distribución como la autoría de material curricular. Esto se refleja en dos de las metas de diseño del proyecto LearningWorks:

- Las herramientas para la distribución de material curricular deben ser las mismas que los alumnos utilizan para llevar registro de sus actividades.

- Las herramientas para la autoría de material curricular deben ser las mismas que utilizan los alumnos para llevar a cabo sus actividades de estudio.

El material desarrollado con LearningWorks debe reunir tres características:

- El material debe consistir de un conjunto de actividades de construcción, comenzando con estudios simples de partes y sus relaciones, y construyendo a partir de ellas proyectos en gran escala de simulaciones complejas.
- Algunas de las actividades de construcción deberían se llevadas a cabo por grupos de estudiantes trabajando en conjunto para planear, escribir y probar soluciones.
- Los grupos de estudiantes deberían formarse partiendo de cursos existentes o por estudiantes que se reúnen en Internet. La formación de equipos debe llevarse a cabo como una actividad más utilizando como medio Internet.

Aunque el sistema debe proveer un conjunto completo de herramientas para el desarrollo de software, este puede diferir de aquel provisto a equipos profesionales de desarrolladores. La mayoría de los sistemas orientados a objetos soportan exploradores (browsers) de clases y métodos integrados y depuradores que utilizan complejas combinaciones de selecciones para definir la semántica de la información mostrada. En contraste, LearningWorks ofrece herramientas separadas para ver los distintos aspectos de la definición de los objetos. El catálogo de herramientas incluye vistas simples de el rol de un objeto en un sistema así como vistas mas complejas de la implementación del objeto. Un autor puede seleccionar las vistas mas apropiadas para el nivel de sus estudiantes.

LearningWorks está pensado como un sistema de aprendizaje de nivel introductorio y apunta a hacer un uso atractivo de gráficos 2D y 3D y de sus capacidades de construcción en mundos virtuales. LearningWorks es como un parque de construcción.

4.2 LearningBooks

Toda la información y actividades de LearningWorks es accesible a través de LearningBooks (libros de LearningWorks), ventanas en la pantalla que simulan la estructura de un libro con secciones y páginas. Las páginas contienen aplicaciones de actividades con las que uno interactúa con el objetivo de explorar los varios tópicos del tema de estudio. Las páginas también pueden contener herramientas para la programación en Smalltalk para explorar la definición de objetos existentes o crear nuevas. Una sección puede tener un tema, que es un conjunto de componentes interactivas que se mantienen en

todas las páginas de la sección. El tema de una sección provee una forma de definir y elegir valores compartidos.

Un LearningBook contiene el software (los objetos) necesario para ejecutar las aplicaciones que aparecen en sus páginas. Dicho software bien puede ser definido por otro LearningBook e importado por este; o ser definido localmente para que pueda ser importado por otros LearningBooks. Este software es utilizado en parte como soporte de las aplicaciones en el libro y en parte para que el alumno lleve a cabo sus actividades de programación. La especificación de un libro indica:

- Qué otros libros incluye importando consecuentemente las definiciones de objetos en ellos;
- Qué objetos nuevos son definidos por el libro;
- Cuáles de estos objetos definidos o importados y cuales métodos de estos objetos son visibles en las herramientas de programación utilizadas por los alumnos.

Haciendo uso de los mecanismo de empaquetamiento de software de LearningWorks los autores pueden construir conjuntos de ejercicios coordinados con acceso controlado a definiciones de objetos preexistentes para ser usados en actividades prácticas. Valiéndose del mecanismo que permite indicar que objetos y que métodos son visibles, el autor controla el acceso a la extremadamente rica librería de Smalltalk (el ambiente de programación profesional sobre en que se construye LearningWorks). Este último mecanismo se llama Visión. La declaración de visión se lleva a cabo tanto a nivel de libro como a nivel de sección.

LearningWorks provee cuatro clases de libros: de actividad, de inspección, de depuración y de autoría. Cada uno tiene un propósito diferente.

Libro de actividad

Contiene la definición de un contexto en el cual el estudiante explora tópicos curriculares incluyendo software de programación. Específicamente define un conjunto de tareas para ser llevadas a cabo por el alumno y provee un mecanismo mediante el cual el alumno desarrolla y mantiene los ejercicios y proyectos. Para programar, este libro tiene herramientas de documentación, prueba e implementación de definiciones, ya sea que estén incluidas en el libro o sean importadas desde otros libros.

Libro de inspección	Presenta información detallada acerca de la estructura de los objetos y las aplicaciones. Típicamente se accede al mismo oprimiendo el botón "Watch" de un libro.
Libro de depuración	Presenta información detallada acerca de la estructura de una secuencia de mensajes entre objetos interrumpida. La única información presentada (clases y métodos) es aquella que se encuentra dentro de la *visión* del libro del cual se inició la interrupción.
Libro de autoría	Presenta opciones de edición para la creación y modificación de LearningBooks, temas de sección, y actividades de páginas.

Un LearningBook puede grabarse y así ser compartido con otros alumnos o tutores. Cuando una actividad de desarrollo de software debe llevarse a cabo cooperativamente por varios alumnos que forman un equipo, cada alumno puede trabajar en un libro separado y luego acceder el trabajo de sus compañeros incluyendo sus libros y haciendo copias de sus páginas.

Como se muestra en las Ilustraciones 4-1, 4-2 y 4-3, un libro puede presentar un proyecto acerca de la manipulación de formas que pueden ser animadas en pantalla para explorar el concepto de sistemas con interface común.

Una sección del libro puede ser dedicada a usar figuras existentes para aprender como pedirles que se muestren en diferentes tamaños, posiciones, y colores (Ilustración 4-1).

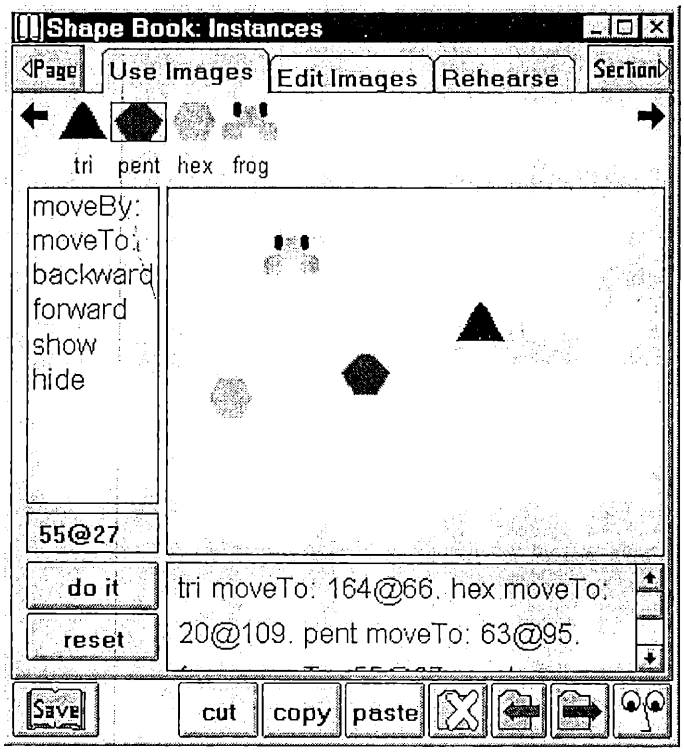
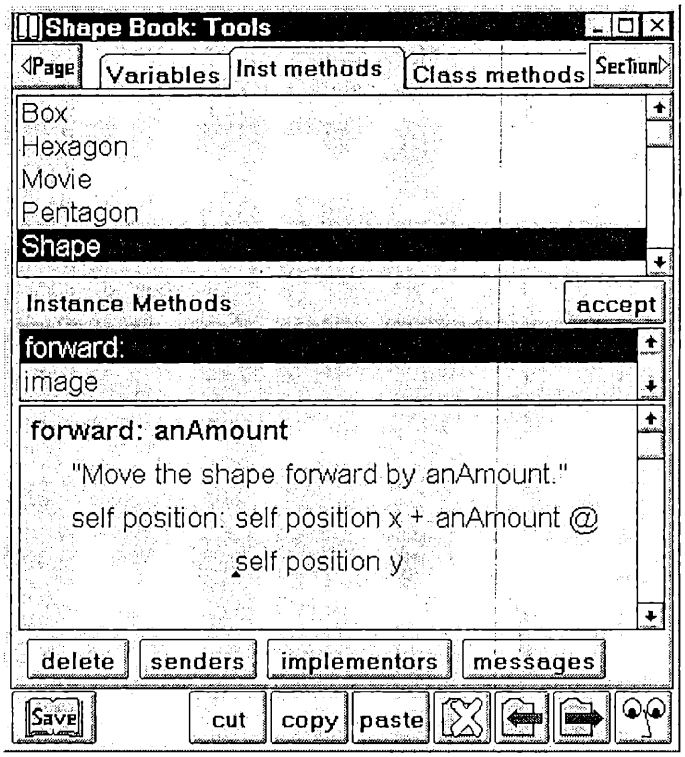


Ilustración 4-1



Otra sección del libro puede ofrecer acceso a la definición Smalltalk de estas formas (Ilustración 4-2).

Ilustración 4-2

Puede también incluir páginas en las cuales el alumno puede escribir definiciones Smalltalk que extienden o crean nuevas formas (Ilustración 4-3)

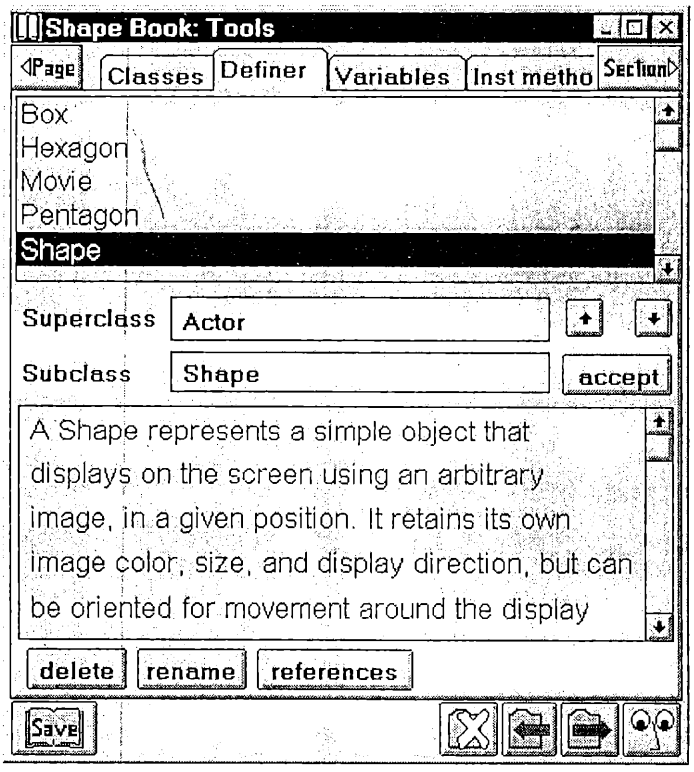


Ilustración 4-3

Las páginas de un libro pueden ser copiadas, separadas del libro y a veces borradas. Insertar una pagina en un libro es hacer una copia de una pagina seleccionada y agregarla al libro como página definida por el usuario. De este modo, el alumno puede llevar a cabo sus actividades de múltiples maneras. Una página puede ser arrancada del libro lo que significa que será colocada en una ventana separada del escritorio. De esta forma los alumnos que dispongan de una pantalla lo suficientemente grande podrán observar múltiples paginas a la vez.

4.3 Cursos de LearningWorks

Los LearningBooks constituyen en mecanismo fundamental de organización de material didáctico en LearningWorks. Los mismos se organizan en cursos que representan las distintas unidades temáticas, proveen acceso centralizado al material de curso e indican aspectos particulares de la configuración del ambiente necesarios para el funcionamientos de sus libros. Los cursos permiten al docente distribuir el material de estudio en unidades cohesivas de fácil acceso. Para el alumno, los cursos representan la posibilidad de llevar cuenta de sus actividades permitiéndole mantener copias privadas de sus libros sin perder las originales.

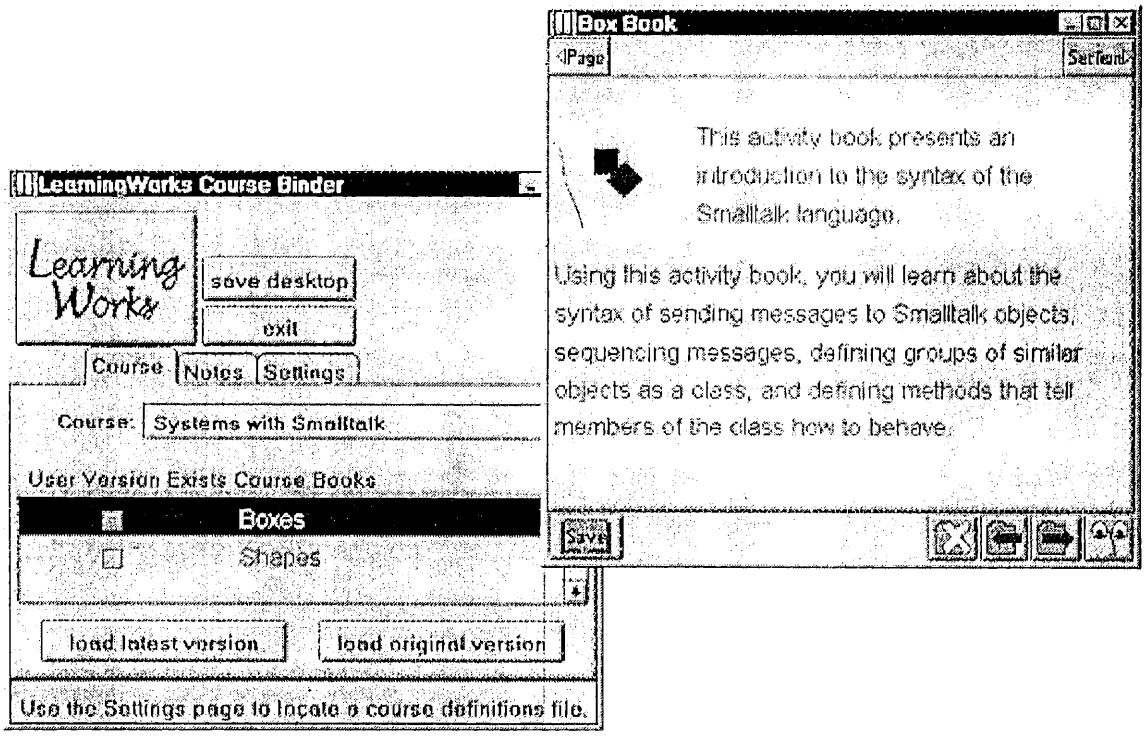


Ilustración 4-4

Desde LearningWorks, los usuarios acceden a cursos por medio del CourseBinder. Esta herramienta permite cargar cursos y alcanzar tanto las versiones originales de sus libros como a las versiones de usuario. La Ilustración 4-4 muestra el CourseBinder junto con un libro recién abierto. La funcionalidad provista por el CourseBinder incluye:

- Ver y abrir los libros que el autor incluyo en un curso.
- Ver y abrir versiones de usuario de los libros, que pueden diferir de las originales dado que incluyen los cambios introducidos en sesiones anteriores.
- Leer los comentarios del cursos escritos por el autor e incluir los propios.
- Revisar o actualizar la configuración del curso (directorios, archivos, etc).
- Grabar una imagen del escritorio de LearningWorks.
- Salir del sistema.

4.4 Versiones de Usuario y Dependencias entre LearningBooks

Todas las actividades que el alumno lleva a cabo se desarrollan en el contexto de libros. Algunas de sus prácticas modifican la estructura propia del libro agregando y eliminando páginas. Otras extienden y modifican el modelo de objetos que soporta al micro-mundo que explora. Un libro resume el trabajo de un alumno en torno a un determinado contenido y sirve como mecanismo para que este pueda compartir sus experiencias con sus pares y sus profesores. Con este fin los libros implementan un mecanismo que permite que todos los cambios introducidos por el alumno se almacenen de forma segura. En la Ilustración 4-4 puede verse el botón **Save** que permite al alumno generar una copia propia (a la que normalmente nos referimos como versión de usuario) del libro en el que trabaja donde se almacenan todos sus cambios.

El formato elegido para representar físicamente los libros en disco se compone de un único archivo por libro. Esto simplifica el proceso de distribución y recolección de libros. Al mismo tiempo, la combinación de este mecanismo con la posibilidad de importar libros desde otros, permite que los alumnos experimenten el desarrollo de software en equipo. Con esta simple funcionalidad los alumnos puede explorar las técnicas de asignación de responsabilidades de desarrollo a los distintos miembros y su posterior integración y prueba.

La generación de versiones de usuarios para los libros del alumno introduce variantes en las estructuras de los cursos. Cuando un curso es originalmente definido se determinan las dependencias

entre sus libros y se proveen versiones originales de cada uno de ellos. Con el paso del tiempo el alumno tendrá sus propias versiones para algunos de ellos. La integración de estas versiones en el curso se lleva a cabo de forma automática durante la carga. Al detectarse la carga de un libro para el que se han definido prerequisitos (por medio de la propiedad use del mismo) se procede a la carga automática de la última versión de cada uno de ellos. De esta forma el alumno puede extender de forma incremental el micro-mundo y entender el impacto que los cambios que introduce tienen en desarrollos futuros.

4.4 LearningWorks en el Mundo

La combinación del soporte para el aprendizaje de tecnología de objetos por medio de micro-mundos con las características particulares de LearningWorks para la distribución y recolección de material educativo hacen del mismo una herramienta mas que interesante. Desde su lanzamiento en 1996, cada vez son mas las universidades que eligen LearningWorks como soporte para sus cursos de objetos. El punto máximo en la utilización del ambiente se consiguió durante el año 1998 cuando LearningWorks fue elegido para el curso de educación a distancia M206 "Computing: An Object Oriented Approach" de la Universidad Abierta del reino Unido. Durante ese año lectivo 5.000 alumnos tomaron el curso cuyas actividades se basaban principalmente en el desarrollo de aplicaciones con LearningWorks. La experiencia de la Universidad Abierta atrajo a otros interesados contando para este año con 600 alumnos en Singapur y con la posibilidad de el lanzamiento de cursos en la Universidad Estatal de Florida.

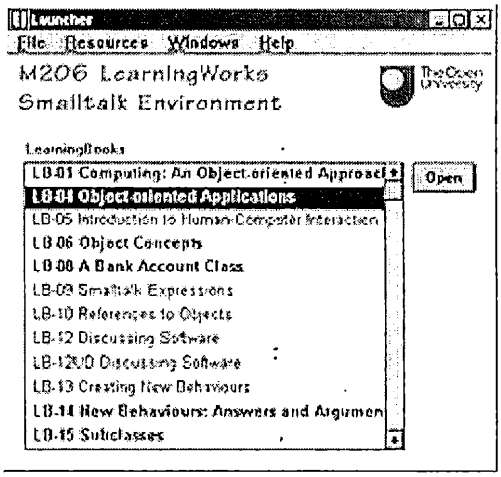


Ilustración 4-5a

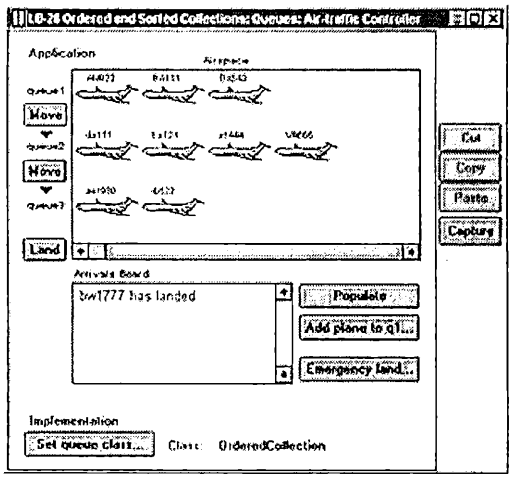


Ilustración 4-5b

La experiencia con la Universidad Abierta fue de gran importancia para la evolución de LearningWorks ya que no solo lo lanzo a la fama sino que se constituyo en una importante fuente de nuevos requerimientos (algunos de los cuales son enfocados en este trabajo).

La Ilustración 4-5a muestra uno de los libros desarrollados para el curso M206 donde se combinan técnicas de hipertexto. La Ilustración 4-5b muestra imágenes del libro para la exploración del micro-mundo de las aeronaves también desarrollado para dicho curso.

4.5 Referencias

[1] Goldberg, Adele, Abell, Steve, y Leibs, David. The LearningWorks Development and Delivery Framework, Communications of the ACM, October, 1997

[2] Goldberg, Adele. OOPSLA 95 Keynote, Addendum to the Proceedings of the ACM OOPSLA Conference, 1995.

[3] Woodman, Mark, Davies, Gordon, Holland, Simon. The Joy of Sotware - Starting with Objects. Proceedings de SIGCSE'96 (88-92). ACM PRESS, 1996. ISSN 0097-8418

[4] Woodman, Mark. Starting Objects with Smalltalk. JOOP Magazine, Ocubre 1999.

Capítulo 5

El Framework de LearningWorks

LearningWorks es también el nombre del framework que permite las construcción de cursos y libros. Este es un framework caja blanca implementado en Smalltalk.

5.1 Construcción de LearningBooks

El framework de LearningWorks hace obvia distinción entre la estructura de un determinado punto o nodo de un libro y la aplicación que ejecuta para dicho nodo. Las aplicaciones se caracterizan por tener un modelo del dominio y una interface de usuario para interactuar con este modelo. En terminología de objetos, las interfaces son el código que maneja la interacción con el usuario y la especificación de la representación gráfica. Para especificar un libro, el autor declara una estructura jerárquica de secciones y páginas dentro de las secciones. El autor indica, para cada nodo en la jerarquía cual es la clase de la cual se creará la interface (*Interface)* y la especificación en esa clase que será utilizada (*Spec)*. Usualmente utilizamos el término *gluecode* para referirnos a la colección de clases de interacción con el usuario. El gluecode hace uso del modelo de objetos del dominio.

El modelo de implementación de un LearningBook utiliza dos estructuras de árbol paralelas, una, con el libro como raíz, para la organización de secciones y páginas y otra para organizar el gluecode. La superclase de cada nodo en el árbol es LwNode y la de cada clase en el gluecode es LwApplicationModel. Cuando un libro es creado, su estructura de LwNodes indica que cosa debe construirse. Una vez construido, la estructura de LwApplicationModels representa las clases de la aplicación pero acude instancias de los correspondientes LwNodes en busca de información acerca del contexto de ejecución local.

La información de contexto local es almacenada en dos variables locals y content de los LwNodes. Cada vez que la interface de usuario ofrece al usuario evaluar expresiones Smalltalk, las variables utilizadas en dichas expresiones y sus valores se recuperan y almacenan de/en locals del

LwNode involucrado. Para algunas actividades el usuario puede declarar nuevas variables locales a la página las que se también almacenan en locals. Similarmente, cualquier otra información sobre la página puede mantenerse en el diccionario contents. La información en locals y content persiste cuando el libro es cerrado y vuelto a abrir o cuando se cambia de página.

La ilustración 5-1 muestra una página de un libro donde se ha definido la variable local localFigure (en el panel de la esquina inferior derecha). Esta es utilizada en la evaluación de expresiones Smalltalk (panel de la esquina inferior izquierda). La evaluación de las expresiones afecta la animación en el panel superior.

Tanto la variable como su valor serán almacenados en el libro.

La necesidad de definición de variables locales es detectada de forma automática al evaluar expresiones.

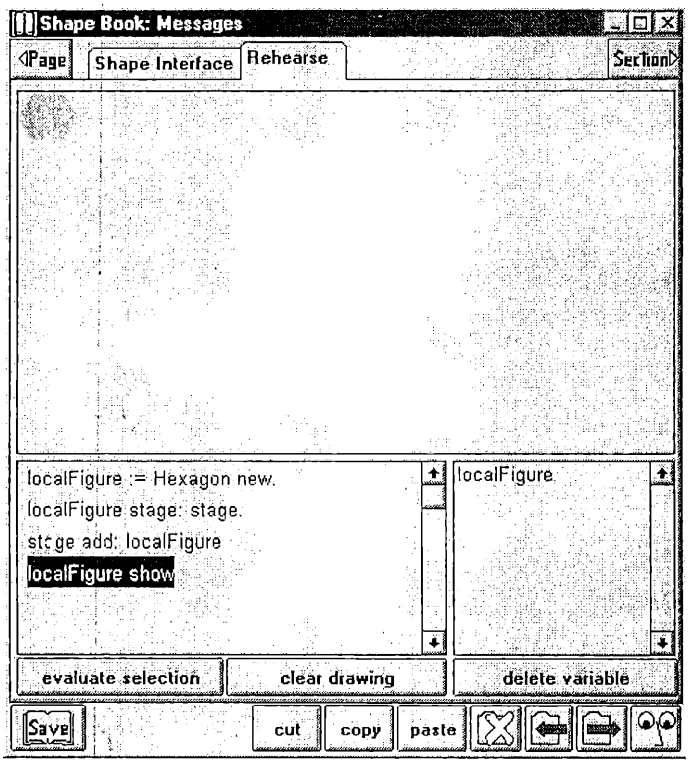


Ilustración 5-1

El autor define la estructura de un libro utilizando la herramienta de estructura, que es provista como una página en el libro de autores de LearningWorks. La herramienta de estructura genera una definición declarativa del libro que fue inspirada por VRML. La Ilustración 5-2 exhibe una definición simplificada del libro de las formas cuyas páginas se muestran en este y anteriores capítulos. El libro en si es implementado, por una clase LearningWorks llamada LwBookApplicationModel. El libro tiene una página de título que es una instancia de la clase del gluecode llamada ShapeTitlePage, indicando que la especificación de su presentación es bookSpec. Las clases ShapePage1, ShapePage2, ShapePage3, ImageEditPage y RehearsePage son también clases del gluecode, todas ellas subclases de LwApplicationModel. La primer sección (etiquetada "Messages") es implementada por la clase LwSectionApplicationModel. Tiene tres páginas. La primera es una página de título, la segunda una pagina instancia de la clase ShapePage1 y la tercera una página instancia de la clase RehearsePage.

Ambas paginas de actividades (la segunda y tercera) mantendrán el estado y propiedades de su interface entre cambios de página dado que para ellas se ha indicado retainInterface como verdadero.

```
Book {
  name 'Shapes'
  use ['Boxes']
  label 'Shape Book'
  interface 'LwBookApplicationModel'
  spec 'windowSpec'
  vision Vision {       define [   'Hexagon' 'Pentagon' 'Triangle']
                        include ['Box' 'Movie' 'Shape']}
  title Title {         label 'Shapes Book Title'
                        interface 'NewLwShapeTitlePage'
                        spec 'bookSpec'}
  elements [            Section {
                        label 'Messages'
                        interface 'LwSectionApplicationModel'
                        spec 'windowSpec'
                        title Title {       label 'Section Title'
                                            interface 'NewLwShapeTitlePage'
                                            spec 'shapeSectionSpec'}
                        elements [          Page {
                                            label 'Shape Interface'
                                            interface 'NewLwShapePage1'
                                            spec 'windowSpec'
                                            retainInterface true}
                                            Page {label 'Rehearse'
                                            interface 'NewLwRehearsePage'
                                            spec 'rehearseSpec'
                                            retainInterface true}]}
                        Section {
                        label 'Instances'
                        interface 'NewLwisImage'
                        spec 'windowSpec'
                        retainInterface true
                        title Title {       label 'Section Title'
                                            interface 'NewLwShapeTitlePage'
                                            spec 'imageSectionSpec'}
                        elements [          Page {label 'Use Images'
                                            interface 'NewLwShapePage2'
                                            spec 'windowSpec'
                                            retainInterface true}
                                            Page {label 'Edit Images'
                                            interface 'NewLwImageEditPage'
                                            spec 'windowSpec'
                                            retainInterface true}
                                            Page {label 'Rehearse'
                                            interface 'NewLwShapePage3'
                                            spec 'rehearseSpec'
                                            retainInterface true}]}]}]}
```

Ilustración 5-2

La segunda sección tiene tres páginas, una para manipular figuras (ShapePage2), una para editar imágenes (ImageEditPage) y finalmente una página de experimentación (ShapePage3).

Para este libro también se declara visión. El libro de figuras utiliza (como lo indica el atributo use de su definición) las clases de modelo definidas en el libro Boxes, otro de los libros ejemplo de LearningWorks. Las dos clases definidas en ese libro son Box y Movie y por estar indicadas en el atributo include son visibles dentro del libro de las formas. Tres nuevas clases de modelo son definidas por el libro de las formas (como lo indica el atributo define) y por eso visibles en él: Hexagon, Pentagon y Triangle. Las definiciones de estas clases se almacenan junto con el libro.

En el archivo de definición del libro, en nombre que define la interface puede ser un nombre de clase que representa su ApplicationModel o puede ser una clave que se resuelve mandando a la clase de la sección el mensaje resolveKey:. La implementación por defecto de este método envía la clave como un mensaje a la sección misma haciendo que efectivamente se resuelva en tiempo de ejecución que ApplicationModel utilizar. El enfoque de resolución por claves da al autor mayor flexibilidad para construir definiciones de libros ya que independiza la misma de las clases que implementan las aplicaciones. El mismo mecanismo funciona para las secciones siendo que la clase es resuelta por el ApplicationModel del libro.

5.2 Clases de Interfaces de LearningWorks

El propósito de la clase abstracta LwApplicationModel es manejar en enganche entre una aplicación y un LwNode en la estructura de árbol del libro, permitiendo que la aplicación sea accesible en una ventana fuera de la estructura del libro. LwApplicationModel provee el protocolo básico para manejar el contexto de un libro, secciones y páginas. Todas las aplicaciones de LearningWorks serán subclases de LwApplicationModel. Mas específicamente, todas las clases que representen libros serán subclases de LwBookApplicationModel, las que representen secciones lo serán de LwSectionApplicationModel y las páginas de actividad de LwNodeApplicationModel.

La Tabla 5-1 detalla los mensajes que componen el protocolo general de interface de LearningWorks. La mayoría de estos mensajes son enviados en respuesta a eventos generados por el usuario. Los mensajes en la tabla representan los enganches para especializar el framework. Cada uno de ellos puede ser reimplementado en una subclase para crear un libro, sección o página especializada.

Evento	Mensaje	Comentarios
agregar secciones y páginas a un libro	getReadyForActivationInBookContext: aBookContext	Prepararse para traer un libro; sección o página a la vida pero cuya interface aún no esta lista. El objeto que representa el contexto del libro (bookContext) da acceso al nodo del libro y puede dar valores iniciales a cualquier contents y local almacenado.
	youHaveBeenActivatedInBookContext: aBookContext	La interface ya ha sido construida y preparada. Tal vez hay algunas acciones adicionales para llevar a cabo antes de dar control al usuario
Cambiar de secciones y páginas en un libro	getReadyForDeactivationInBookContext: aBookContext	La interface del libro todavía esta dibujada (eso significa que el libro todavía se ve). Llevar a cabo lo que sea necesario antes de que la interface de la página sea desactivada. Este mensaje viene seguido de un mensaje release.
Despegar una página	attemptDetachInContext: aBookContext	Despegar una página del libro para utilizarla por separado. No todas permiten ser utilizadas fuera de un libro.

Cerrar una página despegada	attemptReatachInContext: aBookContext	Agrega una pagina nuevamente al libro. La página será ubicada en la misma posición relativa al libro de la que fue despegada.
Comando Watch de inspección	attemptWatchInContext: aBookContext	Crear un libro inspector sobre una variable de una página o sección.
Cerrar un libro inspector	notificationOfCloseInInspectorBookContext: anInspectorBookContext	Cerrar un libro de inspección
Grabar el libro	getReadyForSaveInContext: aBookContext	De ser apropiado, actualizar locals y contents.
Cerrar el libro	release	También puede ser invocado cuando una página no se usa más.

Tabla 5-1

5.3 Construcción de Cursos de LearningWorks

Los cursos de LearningWorks se definen con la misma herramienta de estructura que se utiliza para construir LearningBooks. El libro de autores provee una página especial para declarar la estructura del curso.

```
Course {    name 'Systems with Smalltalk'
            directory '..\Courses\intro\'
            userDirectory '..\Courses\user\'
            glueCode [ 'BoxBk.bos' 'ShapeBk.bos' 'TurtleBk.bos']
            books [   BookInfo {name 'Boxes'
                                label  'Boxes'
                                file 'boxes.lw'}
                      BookInfo {name 'Shapes'
                                label  'Shapes'
                                file 'shapes.lw'}
                      BookInfo {name 'Turtles'
                                label  'Turtles'
                                file 'turtles.lw'}
                      startExpression "LwCourseBinder open"}
```

Ilustración 5-3

Así como para la construcción de libros existe un lenguaje declarativo de definición (BDL, por Book Definition Language), para la construcción de cursos existe otro (CDL, por Course Definition Languaje).

La Ilustración 5-4 muestra la herramienta de estructura con la definición de un curso introductorio a la programación en Smalltalk. La Ilustración 5-3 muestra el archivo de especificación generado por la misma.

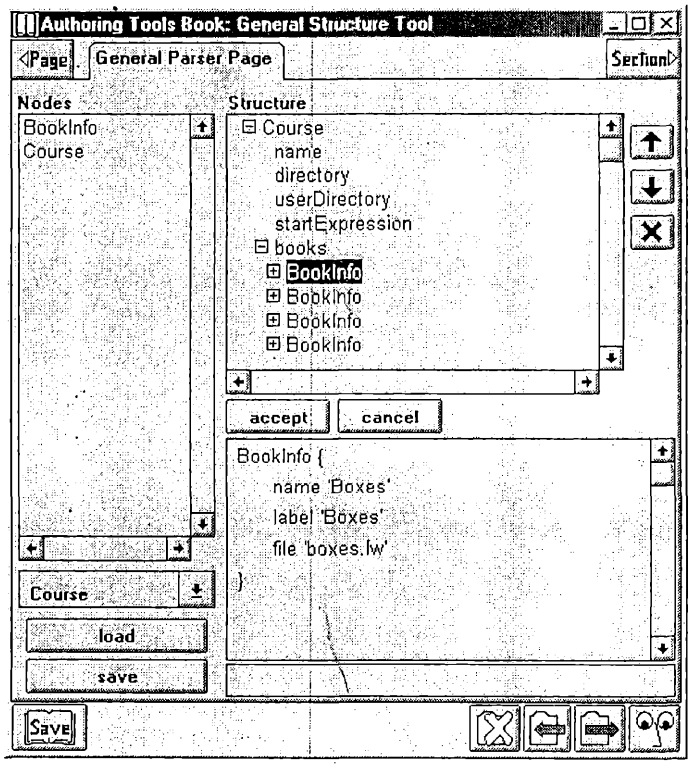


Ilustración 5-4

La descripción de un curso esta principalmente compuesta por una indicación de los directorios de trabajo (atributos directory y userDirectory), una descripción de los libros que lo componen (atributo books) y un detalle de los archivos que contienen las definiciones de las clases de interface de aplicación de dichos libros (atributo gluecode). Esta información, se utiliza durante la carga del curso para chequear que todos los archivos necesarios se encuentran disponibles.

5.4 Framework de Programación - Herramientas

El lenguaje de programación en LearningWorks es Smalltalk. Las páginas de un LearningBook pueden contener herramientas que permitan escribir definiciones Smalltalk. El Framework de

Programación de LearningWorks da soporte a la filosofía Smalltalk al proveer herramientas de visualización y búsqueda para entender la estructura subyacente de un sistema de software. El framework no fuerza a los autores de libros a elegir herramientas particulares sino que ofrece un catálogo de herramientas. Además, los autores tienen control acerca de cuáles definiciones de objetos y cuales aspectos de estas son visibles al usuario.

Las herramientas de desarrollo de sistemas de software utilizando tecnología de objetos consisten, generalmente, en mecanismos de ver y modificar:

1. definiciones de objetos
2. relaciones entre objetos (estructuras estáticas y dinámicas)
   - relaciones definidas estáticamente por la definición del objeto
   - relaciones definidas dinámicamente por las propiedades en tiempo de ejecución de un objeto, incluyendo dependencias
3. implementaciones de objetos
4. comportamiento dinámico de procesos

El usuario de LearningWorks tiene a su disposición un conjunto completo de herramientas para el desarrollo de software con objetos. Algunas de estas herramientas de describen a continuación.

Inspector: Habiendo alguna variable seleccionada, el usuario puede oprimir el botón Watch para abrir el libro de inspección con una página en la que se detalle la estructura interna del objeto apuntado por la variable. En caso de que el libro estuviese abierto se agrega a él una nueva página. El libro de inspección pertenece a la página del libro en la que fue activada. Cuando se cambia de página el inspector se cierra automáticamente. La información en el libro inspector no se almacena.

Referencias: Antes de eliminar o cambiar el nombre de una clase, variable o mensaje se tiene la opción de verificar que no existan referencias al mismo. Automáticamente se crean páginas asociadas al libro que se muestran en ventanas aparte y que detallan referencia a variables, clases o menajes e implementaciones del método seleccionado (si fuera el caso).

Depuradores: La depuración que no es otra cosa que la inspección de un proceso activo se lleva a cabo con el libro de depuración. Cualquiera sea la razón que causa que un proceso se interrumpa se le da la opción al usuario de continuar o de explorar los detalles de la interrupción. El libro de depuración de LearningWorks se ve afectado por la definición de visión del nodo donde se produjo la interrupción.

Exploración: Los libros de inspección, referencias y depuración son herramientas standard de LearningWorks. El autor no debe llevar a cabo ninguna configuración especial para usarlas. Por otro lado cuenta con un conjunto de páginas que pueden ensamblarse en libros para la exploración de definiciones Smalltalk. Estas páginas implementan la funcionalidad básica de un explorador de clases Smalltalk y se ven afectadas por la visión del nodo al que representan. Para incluirlas en la definición de un libro solo hay que utilizar la clase LwBrowserInterface como atributo interface de la sección y seguir las indicaciones en la tabla 5-2. La misma describe que claves utilizar como interface y especificación de la aplicación de cada página de programación.

Tipo de herramienta	Atributo interface	Atributo Spec
Comentario de clase y jerarquía. Solo lectura	classesInterface	classesSpec
Comentario de clase. Solo lectura	classesInterface	classesCommentSpec
Definición de clases	definerInterface	definerSpec
Definición de variables	variablesInterface	variablesSpec
Definición de métodos de instancia	instanceMethodsInterface	instanceMethodSpec
Definición de métodos de clase	classMethodsInterface	classMethodSpec

Tabla 5.2

5.5 Control de Visión

Para el autor el libro es un paquete de software que define, importa y exporta clases y métodos. Cualquier libro puede importar clases y métodos de otros libros. Todo libro incluye una definición de lo que su usuario puede ver a través de las herramientas de programación sin importar que clases y métodos se utilicen para la construcción del libro y sus aplicaciones. La idea es que la visión se va modificando en las distintas secciones par ir revelando al usuario cada vez mas detalles.

La visión de libros y secciones se modela con instancias de la clase Vision. Esta clase representa una estructura que mantiene información acerca de los requerimientos de visibilidad declarados por el autor. Este mecanismo asume:

- Las clases y los métodos importados de otros libros son invisibles sin importar la definición de visión de ellos.
- Cualquier nueva clase o mensaje definido dentro de un libro es visible en ese libro salvo que sea hecho invisible de forma explícita.
- Cuando una clase se elimina del sistema no aparece en ningún lado. Si se carga otro libro que incluye esa clase, ella sigue siendo visible en el libro del que se había eliminado.
- Cuando la estructura de una clase (superclase y subclases) cambia, las declaraciones de visión no se ven afectadas.

Las construcciones del lenguaje de definición de libros (BDL) que permiten la especificación de visión son los VisionNode, VisionRecord y SelectNode. Combinando estas construcciones se consiguen las distintas variaciones en la visión de un libro.

VisionNode: un VisionNode mantiene tres atributos, representando definiciones, inclusiones y exclusiones. Los valores para estos atributos son colecciones eventualmente vacías de Strings o SelectNodes. Cada uno de estos elementos representa una o varias clases y un conjunto de sus métodos de instancia y clase. De tratarse de un String, tendrá el formato Patrón1.Patrón2.Patrón3. donde Patrón1 representa nombres de clase, Patrón2 nombres de métodos de instancia y Patrón3 nombres de métodos de clase, todos con la posibilidad de contener comodines (*). Los SelectNode cumplen una función similar pero son más flexibles ya que permiten enumerar uno a uno los mensajes a incluir. El atributo define (definición) de la visión indica que elementos fueron definidos por el libro y se almacenan en él. El atributo include (inclusión) indica que elementos pueden verse. El atributo exclude (exclusión) indica que elementos no pueden verse.

```
Vision { include [
        'Movie'
        'Box.*.p*'
        Select {
                name 'Turtle'
                instance ['backward:'
                        VisionRecord { name 'forward:' restrictions 2 }]
                meta ['*']}]}
```

Ilustración 5-5

RecordNode: los RecordNode se combinan con los SelectNode para detallar cuales van a ser los permisos de acceso del usuario a un determinado método. Un RecordNode indica un nombre de mensaje (que puede incluir comodines) y un valor de restricción. Los únicos valores posibles de

restricción son 0 (acceso total) y 2 (solo ver el nombre y comentario). La Ilustración 5-5 muestra un ejemplo de definición de visión.

5.6 Referencias

[1] Carey, Rikk, y Bell, Gavin. The Annotated VRML 2.0 Reference Manual, Addison-Wesley: Reading, 1997.

[2] Goldberg, Adele. *Smalltalk-80*: The Interactive Programming Environment, Addison-Wesley: Reading, 1984.

[3] Goldberg, Adele, Robson, David. Smalltalk-80. The Language and its Implementation. Addison Wesley, 1983. ISBN 0-201-11371-6

[4] Goldberg, Adele. OOPSLA 95 Keynote, Addendum to the Proceedings of the ACM OOPSLA Conference, 1995.

[5] Goldberg, Adele, Abell, Steve, y Leibs, David. The LearningWorks Development and Delivery Framework, Communications of the ACM, October, 1997

Capítulo 6

Modelo de Objetos de Smalltalk 80

Smalltalk se destaca de todo otro lenguaje orientado a objetos conocido por contar con un modelo de objetos completo de si mismo. Este modelo de objetos determina el comportamiento total del ambiente. Cuenta, entre otras, con clases para modelar el compilador, las clases, los métodos, el espacio de nombres del ambiente, el explorador de clases, el inspector de objetos y el depurador. Esto permite desarrollar nuevas versiones del ambiente con herramientas o características de lenguajes específicas con solo programar Smalltalk. Esta característica es normalmente conocida con el nombre de reflexión y permitió la construcción de LearningWorks sobre VisualWorks Smalltalk.

En la construcción de herramientas de programación, el foco de desarrollo se dirige a las clases, los métodos y el espacio de nombres (también conocido como repositorio global de clases). Cada vez que se define una nueva clase se crea una instancia de la clase Class y se agrega al repositorio de clases. Cada vez que se define un método, se crea una instancia de la clase CompiledMethod, que se asocia a la clase a la que pertenece.

Este capítulo describe brevemente las principales responsabilidades y el funcionamiento de las clases, los métodos y el repositorio de clases. Este conocimiento es necesario para poder entender como se construyen las herramientas de programación. Este modelo de objetos es extendido en el Capitulo 8 para completar la funcionalidad de las herramientas con el objetivo de soportar reversión de cambios y control de visión.

6.1 Clases y el Repositorio de Clases

La construcción de aplicaciones en objetos se centra en la definición de clases. En Smalltalk esto implica crear nuevos objetos. Cada clase esta modelada por una instancia de una clase particular, generada automáticamente y subclase de la clase Metaclass. Aunque suene complicado es transparente para el programador.

6.1.1 Definición de Clases

Todo lo que debe hacerse para definir una nueva clase es enviar un mensaje a aquella que va a ser su superclase. Los parámetros de este mensaje indican nombre de al nueva clase, nombres de variables de instancia y clase, categoría de la clase (irrelevante en el contexto de LearningWorks).

Cada nueva clase es almacenada en un SystemDictionary al que se referencia globalmente con el nombre de Smalltalk. Los nombres en este diccionarios son utilizados para resolver las referencias a nombres de clases (a globales en general) durante la compilación.

La Ilustración 6.2 muestra este mecanismo para el caso de la creación de la clase VRChannel, subclase de VRRegion. Sus variables de instancia son name y pov y sus variables de clases BoundaryURL y LocaleURL. Como resultado de esta operación, una nueva clase es creada y agregada Smalltalk.

```
VRRegion subclass: #VRChannel
        instanceVariableNames: 'name pov'
        classVariableNames: 'BoundaryURL LocaleURL '
        poolDictionaries: ''
        category: 'VRObjts'
```

Ilustración 6.1

Esta misma operación se repite cada vez que quiere cambiarse la superclase, agregar o quitar variables o cambiar la categoría.

6.1.2 Comentarios de Clase y Variables

En VisualWorks cada clase cuenta con un comentario que determina su objetivo y describe su estructura. Para establecer el comentario de una clase se debe mandar a la misma el mensaje comment: con el texto que representa su comentario como parámetro. El cambio de esta propiedad no afecta de ninguna otra manera a la clase. Así mismo, otros cambios en la clase no afectan su comentario. Para conseguir el comentario de una clase dada se le envía el mensaje comment (sin parámetros)

LearningWorks hace cambios al modelo básico de VisualWorks permitiendo definir comentarios para las variables de una clase. Para comentar una variable de clase o instancia se dispone de los mensajes classVarCommentAt:put: y instVarCommentAt:put:. Ambos reciben como parámetros en nombre de la variable y el texto del comentario. Establecer el comentario de una variable no afecta de otra forma a la clase, sin embargo, los cambios en la estructura de la misma que eliminan variables

pierden sus respectivos comentarios. Para conseguir el comentario de una variable de instancia se envía a la clase el mensaje classVarCommentAt: con el nombre de la variable como parámetro.

6.1.3 Remoción de Clases

Eliminar clases del ambientes es un proceso complicado pero que queda totalmente oculto al programador de herramientas. Se debe verificar que el sistema mantenga la consistencia y luego indicar a la clase misma que debe removerse del ambiente mandándole el mensaje removeFromSystem. En respuesta no solo eliminara la referencia a ella que existe en el repositorio sino que también limpiará toda relación que exista entre ella su superclase y sus subclases (de hecho, estas últimas son también eliminadas)

Al eliminar una clase del sistema, se eliminan con ella todas sus subclases y todos sus métodos. Todas las propiedades de la clase se pierden.

6.2 Métodos

La funcionalidad de las clases esta dada principalmente por sus métodos de clase e instancia que determinan su propio comportamiento y el de sus instancias respectivamente. La definición de métodos se inicia con la compilación del código fuente que los define. Un método se identifica por un selector (o nombre) y encapsula el código compilado que lo implementa. También cuenta con mecanismos para recuperar el código fuente que le dio origen.

Cada clase cuenta con un diccionario de métodos que asocia los selectores a los métodos compilados. Este diccionario evoluciona con los cambios que el programador hace a la clase y es también utilizado por el mecanismo de búsqueda de métodos.

6.2.1 Compilación, Recuperación y Eliminación de Métodos

Desde el punto de vista del programador la compilación de un método se consigue enviando a la clase, alguna de las variantes del método compile:classified: que recibe como parámetros el texto del método y la categoría en la que este debe ubicarse (LearningWorks no hace caso a la organización por categorías). Como resultado, si el método pudo compilarse satisfactoriamente, se instala en el diccionario de métodos de la clase receptora. De existir un método para dicho selector, se pierde toda información de él.

Las clases cuentan con protocolo que permite acceder a sus métodos. El mensaje selectors retorna los selectores de todos los mensajes implementados por la clase. El mensaje compiledMethodAt: recibe como parámetro un selector y retorna el método compilado correspondiente. Para recuperar el código fuente de un método se dispone del mensaje sourceCodeAt: que recibe como parámetro un selector y retorna el código fuente del método.

Para eliminar un método solo debe enviarse a la clase en mensaje removeSelector: con el selector del método como parámetro.

6.3 Referencias

[1] Goldberg, Adele, Robson, David. Smalltalk-80. The Language and its Implementation. Addison Wesley, 1983. ISBN 0-201-11371-6

Capítulo 7

Limitaciones de LearningWorks

LearningWorks se presenta como un ambiente introductorio para la exploración de conceptos de construcción de sistemas de software con objetos. Su fuente de inspiración principal son los micro-mundos, un modelo de aprendizaje constructivista basado en la exploración y modificación de sistemas de objetos simples, familiares e incompletos. Su organización en cursos y libros constituye el mecanismo principal de empaquetamiento, organización y distribución de material curricular. Los libros y cursos proporcionan los objetos del micro-mundo sobre el que se va a desarrollar las actividades así también como las herramientas necesarias para explorarlo y extenderlo.

La actividad fundamental en el aprendizaje con LearningWorks es la programación en Smalltalk. Dado que LearningWorks fue directamente construido sobre VisualWorks Smalltalk (la versión de Smalltalk de Object-Share), provee acceso a toda su funcionalidad y respeta su misma arquitectura. Mientras que LearningWorks es comúnmente descripto como un Smalltalk académico de nivel introductorio y de distribución gratuita, podría asegurarse que es solo una extensión a VisualWorks. De hecho, esta es la base de su implementación. LearningWorks se consigue gracias a las virtudes de reflexión de Smalltalk que permiten que el ambiente evolucione en direcciones que ni siquiera fueron previstas. Las herramientas de programación en LearningWorks y sus mecanismos para el empaquetamiento de software en libros y cursos no son otra cosa que nuevas versiones de sus pares en VisualWorks que fueron optimizadas para servir en un contexto de aprendizaje.

LearningWorks hereda el poder y arquitectura de Smalltalk y lo combina con la teoría de micro-mundos con el objetivo de conseguir un ambiente de aprendizaje óptimo. Sin embargo, a pesar de la gran aceptación del ambiente en el entorno académico todavía presenta signos de inmadurez.

Al momento de efectuado este trabajo, LearningWorks se distribuye en su versión 0.8, que aún cuenta con herramientas en estado de prototipo. El objetivo de los principales involucrados en el desarrollo del ambiente es que el mismo evoluciones en base a el interés, motivación y esfuerzo de la

quienes lo utilizan. Es en ese contexto que se origina este trabajo. El objetivo es extender LearningWorks en tres aspectos en los que se encuentra actualmente limitado: control y reversión de cambios, degradación progresiva del ambiente y configuración de visión y acceso. A continuación de caracterizan en detalle cada uno de estos aspectos.

7.1 Control y Reversión de Cambios

La programación en Smalltalk gira entorno a la creación y modificación de clases. Por su modelo de funcionamiento (descripto en el capítulo 6) mantiene todas las clases y sus métodos en un único repositorio localizado en la imagen virtual de memoria.

De no contarse con alguna herramienta de manejo de código (por ejemplo Envy) el principal mecanismo para dar persistencia a los cambios es grabar la imagen. Este proceso permite a Smalltalk almacenar en disco una copia de la imagen virtual para su posterior utilización. Al grabar la imagen virtual se almacenan con ella las clases y todos los objetos que componen el sistema incluyendo las propias herramientas de programación que puedan estar utilizándose en ese momento. Si bien este es un mecanismo muy poderoso ya que permite almacenar el estado del escritorio de trabajo es también muy costoso en tiempo y en espacio (en el orden del minuto y mas de 10 Megabytes de espacio).

LearningWorks provee su propio mecanismo de empaquetamiento y almacenamiento de código, los libros. Es por ello que solo se recurre a grabar una copia del escritorio de trabajo raramente y en situaciones muy particulares. Para el usuario promedio, la existencia de esta funcionalidad pasa desapercibida. Cuando es necesario almacenar los cambios efectuados en el contexto de un libro se procede a grabar una nueva versión del mismo. Al cargar un curso, si se dispone de una versión modificada de alguno de sus libros se dará al usuario la posibilidad de abrirlo, permitiéndole acceder a los cambios que llevo a cabo la última vez que trabajó en el.

En un escenario real, los usuarios comienzan LearningWorks y cargan cursos y libros uno tras otro llevando a cabo sus actividades. Mientras exploran el micro-mundo que se les presenta tienen la oportunidad de extenderlo, cambiarlo y posiblemente de hacer modificaciones que pongan en peligro la integridad del mismo. Todo esto es parte del aprendizaje exploratorio. Cuando el alumno termina con las actividades propuestas por un libro generalmente lo graba (posiblemente para mostrarlo al profesor) y lo cierra. Luego continúa con las actividades en otro libro sin siquiera haber salido de LearningWorks. Aunque decida no grabar un libro, los cambios que efectuados en él permanecen en el ambiente durante toda la sesión. Sin embargo no estarán allí la próxima vez que inicie LearningWorks. Esto nos conduce a una situación indeseable. La falta de mecanismos para la reversión de los cambios

efectuados en un libro una vez que este se cierra hace que el funcionamiento de otros se vuelva impredecible. Esto no solo afecta el funcionamiento del ambiente sino que también atenta contra el proceso de aprendizaje al exponer al usuario a situaciones inconsistentes.

7.2 Degradación Progresiva del Ambiente

Los libros de LearningWorks actúan como paquetes de software. Almacenan objetos, clases y métodos. Cuando un libro se carga todas las clases y métodos que define se instalan en el repositorio de clases. De existir versiones anteriores de ellos, las mismas son reemplazadas por las nuevas sin dejar lugar a revertir este proceso.

Las clases y métodos en un libro forman generalmente parte de la definición del micro-mundo a explorar, sin embargo no hay ninguna restricción al respecto. De hecho el libro podría cambiar definiciones de clases y métodos que afecten el funcionamiento del ambiente mismo. El usuario no siempre es consciente de los cambios que introduce al ambiente cuando carga un libro. En particular, podría darse el caso de un alumno que carga un libro que introduce modificaciones que van mas allá de su percepción (dada por la configuración de visión y herramientas de exploración).

Un ejemplo claro de esta situación se encuentra en el Libro de Visualización. Este extiende LearningWorks con un framework de visualización de objetos en tiempo real no intrusivo. El mismo permite interceptar eventos en el modelo de objetos que luego pueden ser utilizados para la construcción de visualizaciones. Este framework se basa en la introducción de extensiones al meta-modelo de VisualWorks. Las definiciones de clases y métodos son modificadas de manera casi imperceptible aplicando técnicas de programación reflexiva (lightweight classes, wrappers) para permitir capturar eventos de ejecución de métodos y acceso a variables. Estas modificaciones afectan el corazón mismo del ambiente y, de entrar en conflicto con cambios subsiguientes, pueden llevar el ambiente a estados inseguros y de mal funcionamiento. Asimismo este tipo de extensiones producen degradación de performance. Si bien la misma se justifica mientras se hace uso de las capacidades de visualización, no tiene sentido una vez que estas dejan de ser útiles.

Por la actual implementación del mecanismo de carga de libros, LearningWorks no permite que los cambios introducidos se deshagan. Todas las modificaciones introducidas por los libros para soportar su propio funcionamiento se mantienen en el ambiente hasta finalizar la cesión. La carga consecutiva de libros de forma aleatoria genera combinaciones imprevistas de cambios que conducen a la degradación progresiva e inconsciente del ambiente. Esta situación es sin duda indeseable. Como se

indicaba en el inciso 7.1, estos estados del ambiente no solo atentan contra el funcionamiento propio de la herramienta sino que exponen al alumnos a situaciones de aprendizaje no recomendables.

7.3 Configuración de Visión

Uno de los mecanismos fundamentales para la exploración de micro-mundos de LearningWorks es el control de visión. En el aprendizaje con micro-mundos el alumno tiene la oportunidad de inspeccionar, modificar y extender las definiciones de los objetos con el objetivo de llegar a sus propias conclusiones con respecto a las relaciones y responsabilidades de los mismos. Dado que esta es una actividad no guiada, el usuario puede llevar a cabo casi cualquier cambio. El mecanismo de visión permite al docente limitar las capacidades del alumno en función de dos metas:

- evitar que efectúe modificaciones que atenten contra el funcionamiento del ambiente,
- exponer al alumno a situaciones de exploración particulares.

Cuando se desarrollan actividades en ambientes reflexivos abiertos (como Smalltalk) se corre el peligro de introducir modificaciones que conduzcan a la destrucción del mismo. Dado que todas las clases que lo definen, se encuentran disponibles y pueden ser modificadas, se debe tener mucho cuidado al efectuar cambios. La sola modificación equívoca de uno de los métodos de la clase Rectángulo (usada intensivamente para el manejo de ventanas) y el ambiente dejara de funcionar. Conocer el efecto de cada modificación requiere un profundo entendimiento del rol de cada objeto dentro del ambiente. Quien desarrolla actividades Smalltalk conoce estas características del ambiente. Esto permite al autor de LearningBooks prepararlos configurando su visión de forma que permite al alumno la mayor libertad y seguridad posible en sus experiencias de exploración.

En el aprendizaje de objetos se exploran ciertos aspectos de diseño que son fundamentales. Entre ellos la herencia, la composición y la dependencia en los protocolos de los objetos y no en su implementación. Un control de visión flexible permite que el docente exponga al alumno a condiciones de trabajo donde estos conceptos toman mayor importancia. Por ejemplo, podría prepararse el escenario donde el alumno necesita conseguir objetos con propiedades similares a la de una clase existente pero se ve restringido de la capacidad de modificar la misma. Esto lo conducirá automáticamente a considerar el mecanismo de herencia como una alternativa. Otro ejemplo muy interesante se da cuando los alumnos solo tienen acceso a la descripción de comportamiento de los objetos y no a su implementación. Esto refuerza la importancia del encapsulamiento y ocultamiento de implementación.

El mecanismo de control de visión de LearningWorks en su versión 0.8 es limitado en este contexto. En lo que respecta a visión de clases solo se dispone de dos variantes: que sea accesible para el alumno o que no. Por el lado de los métodos se cuenta con mayor flexibilidad, pero aún es pobre. Por cada método puede indicarse si será visible o no, y en caso de serlo si solo se tendrá permiso de lectura de su comentario o si se tendrá acceso total a él.

7.4 Requerimientos y Evaluación de Impacto

Luego de sucesivas discusiones con los autores del ambiente y algunos de sus usuarios finales se llega a la conclusión de que los aspectos mencionados en los puntos 7.1, 7.2 y 7.3 presentan limitaciones importantes del ambiente. Las mismos deben ser estudiadas en detalle y se deben proponer alternativas de implementación acordes para ser incluidas en posteriores versiones de la herramienta.

Para una correcta evaluación de los cambios necesarios y del impacto que los mismos tendrán en la evolución del ambiente se hace necesario un profundo análisis de su implementación y relación con VisualWorks. En particular debe prestarse gran atención al funcionamiento de los ambientes Smalltalk con respecto al manejo de definiciones de clases y métodos. De esto dependerá el éxito en resolver los problemas planteados en las secciones 7.2 y 7.3. También será necesario conocer en profundidad el mecanismo de empaquetamiento y distribución de LearningWorks a fin de extenderlo en pos de resolver las limitaciones expuestas en los incisos 7.1 y 7.2.

En conjunto con las extensiones que decidan llevarse a cabo, se dispone la migración a la versión 3.0 de VisualWorks. Si bien se estima que el cambió puede producirse sin mayores complicaciones, requiere un análisis exhaustivo de la relación entre la funcionalidad de LearningWorks y las nuevas capacidades provistas en VisualWorks 3.0. Algunos de los aspectos mas relevantes son la actualización del mecanismo de Parcels, y la extensión del framework de interfaces para el soporte de eventos en lugar de pooling.

7.5 Referencias

[1] Fernández, Alejandro, Gustavo Rossi et. al. A Learning Environment to Improve Object Oriented Thinking. Publicado en "Educator's Symposium Notes", OOPSLA'98. ACM Press.

[2] Goldberg, Adele, Abell, Steve, y Leibs, David. The LearningWorks Development and Delivery Framework, Communications of the ACM, October, 1997

[3] Goldberg, Adele, Robson, David. Smalltalk-80. The Language and its Implementation. Addison Wesley, 1983. ISBN 0-201-11371-6

[4] Goldberg, Adele. OOPSLA 95 Keynote, Addendum to the Proceedings of the ACM OOPSLA Conference, 1995.

[5] Lalonde, Wilf R, Pugh, John R. Inside Smalltalk: Volume 1. Prentice Hall Engineering, Science & Math, 1990. ISBN 0-13-468414-1

[6] Lalonde, Wilf R, Pugh, John R. Inside Smalltalk: Volume 2. Prentice Hall Engineering, Science & Math, ISBN 0-13-465964-3

[7] Lalonde, Wilf R, Pugh, John R. Smalltalk/V: Practice and Experience. Prentice Hall Engineering, Science & Math, 1993. ISBN 0-13-814039-1

[8] Woodman, Mark, Davies, Gordon, Holland, Simon. The Joy of Sotware - Starting with Objects. Proceedings de SIGCSE'96 (88-92). ACM PRESS, 1996. ISSN 0097-8418

[9] Woodman, Mark. Starting Objects with Smalltalk. JOOP Magazine, Ocubre 1999.

Capítulo 8

Integridad y Visión: un Nuevo Modelo

Los capítulos precedentes exponen el contexto en el cual se lleva a cabo el desarrollo de LearningWorks, detallando sus características más relevantes y haciendo notar tres aspectos en los que se encuentra limitado. Esta sección presenta un modelo de implementación alternativo para superar dichas limitaciones.

Los tres aspectos mencionados en las secciones 7.1, 7.2 y 7.3 se relacionan principalmente a dos áreas de la implementación de LearningWorks bien identificadas. Por un lado se tiene el mecanismo de Smalltalk para la definición y modificación de clases y métodos y la forma en que este se integra en LearningWorks. Por otro lado se tiene el soporte de visión y el uso que hacen del mismo las herramientas de programación LearningWorks. Ambos aspectos entran en contacto en el framework de programación LearningWorks de donde se toma la mayor parte del funcionamiento de los libros de programación. Los cambios propuestos en esta sección derivan, por ende, en un nuevo framework de programación, que afectan también el modelo de definición de visión de libros y por consiguiente al Lenguaje de Definición de Libros (BDL).

Como meta secundaria se busca que los cambios a efectuarse tengan el menor impacto posible en los cursos que ya han sido desarrollados. Esto lleva a pensar en la eventual convivencia de ambas versiones del framework (la original y la extendida para soportar los nuevos requerimientos)

8.1 Extensión del Mecanismo de Introducción de Cambios

La búsqueda de soporte para la reversión de las acciones del usuario y la limitación de la degradación progresiva de ambiente (originada en la carga de libros) se relacionan directamente con los mecanismos primitivos del ambiente para la introducción de cambios (en las definiciones de las clases y sus métodos). En un caso, los cambios son introducidos de manera consciente por el usuario

valiéndose de las herramientas de programación. En otro caso los cambios son introducidos transparentemente durante la carga de los libros.

En ambos casos se llevan a cabo las mismas operaciones las cuales resultan en la aparición de nuevas clases y métodos, la evolución de los existentes y aún su desaparición. Esto sugiere que de ser posible controlarlos o contabilizarlos de alguna manera y de existir un mecanismo para revertirlos podría resolverse ambos problemas de forma análoga. Sin embargo, la información necesaria en cada caso para revertir el proceso se genera y obtiene en situaciones dispares que deben ser analizadas por separado.

En la versión 3.0 de VisualWorks Smalltalk se puede observar una situación similar. Esta herramienta implementa un mecanismo de empaquetamiento de código llamada Parcels. Los parcels son paquetes de software compuestos principalmente por clases y métodos. Durante la carga de un parcel se guardan el lugar seguro las versiones que ya puedan existir de los métodos que van a instalarse y luego se procede a la instalación de los mismos. Durante la descarga del parcel los métodos instalados por él son reemplazados por aquellos que existían con anterioridad, dando lugar así a la reversión de los cambios efectuados. De la misma manera, el sistema de parcels extiende las herramientas de programación Smalltalk para permitir reconocer los cambios que se llevan a cabo y así posibilitar la vuelta atrás. En cada cambio efectuado por el usuario se recuerda de alguna manera el estado original del ambiente para ser posteriormente restaurado.

Este enfoque, si bien es efectivo para VisualWorks, no es lo suficientemente flexible para adaptarse a LearningWorks. Se ve limitado por el tipo de cambios con los que trata y por el tipo de información que empaqueta.

En un ambiente totalmente reflexivo (como es el caso de Smalltalk), podría considerarse una reimplementación del mecanismo de parcels para ser utilizado en un nuevo contexto. Sin embargo, luego de la evaluación del costo de la misma, se toma como alternativa mas viable extender el corriente sistema de empaquetamiento y programación de LearningWorks teniendo, seguramente, en cuenta las dos características observadas en el sistema de parcels: extensión de las herramientas, adaptación del mecanismo de carga de paquetes.

Por consiguiente, las herramientas de programación de Learningworks modeladas como páginas de libros (descriptas en la Sección 5.4) deberán extenderse para mantener información actualizada de las acciones y cambios efectuados con el objetivo de facilitar la vuelta atrás en el momento de la descarga del libro en cuestión. El mecanismo de empaquetamiento y grabación de

libros (que se detalla en la Sección 5.6) así como también la estructura de los archivos generados debe completarse con información que permita determinar al momento de la carga cuales serán las modificaciones a introducir. Esta información se integrará con aquella mantenida por las herramientas de programación lo que posibilitará deshacer todos los cambios de manera integrada y uniforme.

8.1.1 Identificación y Reversión de Cambios

Antes de proceder a estudiar en detalle como modelar, detectar y almacenar los cambios en ambos casos debemos identificarlos. También debe determinarse el mecanismo para revertir cada uno de ellos y la forma en la que se relacionan.

La Tabla 8.1 enumera las operaciones sobre el modelo de objetos soportadas por las herramientas de programación de LearningWorks y las describe en términos generales.

Cambios referentes a clases	**En el modelo subyacente**	**Descripción**
Definición	Creación de clase	Agregar una nueva clase al ambiente indicando su nombre y su superclase. No se determina el comentario de clase ni la descripción de su estructura.
Cambio de nombre	Cambio de nombre de la clase	Cambiar el nombre de una clase existente en el ambiente. No afecta ni la definición de estructura ni el comentario.
Cambio de comentario	Cambio de comentario	Alterar el comentario de una clase existente en el ambiente. No afecta su definición.
Cambio de superclase	Redefinición de clase	Cambiar la superclase de una clase existente en el ambiente. No afecta ni la definición de estructura ni el comentario.
Agregar/Eliminar variable de instancia o clase	Redefinición de clase	Cambiar la estructura interna de la clase y sus instancias agregando o eliminando variables de instancia y clase. No afecta el comentario de la clase. Pierde el comentario de las variables eliminadas y no determina comentario para las nuevas.

Eliminar	Eliminar clase	Eliminar completamente la clase del ambiente.
Cambio de comentario de una variable de clase o instancia	Comentar la variable de clase o instancia	Alterar el comentario de una variable de clase o instancia. No se afecta la definición de la clase ni su comentario.
Cambios referentes a métodos	**En el modelo subyacente**	**Descripción**
Agregar/Modificar un método de instancia o clase	Compilar método	Compilar un nuevo método y agregarlo al diccionario de métodos de la clase o metaclase según corresponda. De existir, se olvida la versión anterior.
Eliminar	Eliminar método	Elimina por completo un método de instancia o clase

Tabla 8-1

La Tabla 8-1 indica también la forma en que cada operación es modelada por operaciones de menor granularidad sobre los objetos del modelo Smalltalk subyacente (basadas en los conceptos cubiertos en el Capítulo 6). Del estudio de cada una de ellas se deriva un mecanismo para deshacer su efecto y por consiguiente el de las operaciones de LearningWorks.

Creación de clase: para la definición de una nueva clase debe indicarse su nombre (que no debe existir en el ambiente) y su superclase. Este proceso genera una nueva entrada en el espacio de nombre global del ambiente para referirse a la misma. ***Revertir*** el proceso de creación de clase implica remover completamente la misma del ambiente.

Redefinición de clase: La redefinición de clase es uno de los procesos más completos y complejos. Se indica una superclase, sus variables de instancia y sus variables de clase. Su definición se altera teniendo en cuenta todos estos parámetros. ***Revertir*** el proceso de definición de clase implica efectuar nuevamente la operación utilizando los parámetros de superclase y variables originales. Los mismos deben ser almacenados con anterioridad.

Cambio de nombre de clase: El nombre de una clase es una propiedad que puede ser alterada valiéndose de los mecanismos adecuados. Los mismos aseguran que se mantenga la consistencia del ambiente. Todo lo que debe indicarse es un nuevo nombre, que no haya sido tomado aún y que

responda a los estándares. Para ***revertir*** el cambio de nombre solo es necesario repetir el proceso con el nombre original. El mismo debe ser guardado con anterioridad para posibilitar esta tarea.

Cambio de comentario de clase: El comentario es otra propiedad de la clase que puede alterase muy simplemente. El único parámetro necesario es el nuevo comentario. El proceso pierde toda referencia al comentario anterior. Un cambio de comentario puede ***revertirse*** repitiendo la operación con el comentario original. El mismo debe ser guardado con anterioridad para posibilitar esta tarea.

Comentar variables: LearningWorks extiende Smalltalk con la posibilidad de agregar comentarios a las variables de instancia y clase. Para esto solo debe indicarse la variable en cuestión y el nuevo comentario. El comentario anterior, de existir, es descartado. Para ***volver atrás*** el comentario de una variable se debe efectuar la misma operación pero utilizando el antiguo comentario que debió ser almacenado con anterioridad.

Eliminar una clase: Eliminar una clase del ambiente es un proceso complejo pero que esta directamente soportado por el modelo reflexivo de Smalltalk. La clase se remueve por completo del ambiente incluyendo todos sus métodos, comentario de clase y comentarios de variables. Para ***revertir*** este proceso se debe crear una copia fiel de la clase. Previamente, debieron almacenarse con todos sus atributos.

Compilar un método: La compilación de un método de clase o instancia se inicia con el código fuente del mismo y culmina con la introducción de un método compilado en el diccionario de métodos de la clase o metaclase según corresponda. El proceso es soportado totalmente por el modelo subyacente y permite determinar cual es el selector del método. El mismo es necesario para proceder a la ***reversión*** de este cambio. Para remover un método del diccionario correspondiente se debe contar con el selector (o firma) del método y la clase a la que pertenece.

Eliminar un método: Smalltalk provee soporte para simplemente eliminar un método de una clase con solo indicar su selector. Revertir este proceso requiere que el método compilado sea instalado nuevamente en la clase. Para esto es necesario recordar el selector de método, su código compilado y su código fuente.

Durante el desarrollo de las actividades en LearningWorks estas 8 operaciones se suceden una tras otra dando forma a nuevos modelos de objetos. El efecto de las mismas puede revertirse si: 1) a cada paso se almacena suficiente información como para deshacer cada operación; 2) se recuerda el

orden en que las operaciones fueron efectuadas 3) se procede a deshacer los cambios de a uno, desde el mas nuevo al mas antiguo.

	Creación de clase	Redefinición de clase	Cambio de nombre	Cambio de comentario	Comentar variable	Eliminar una clase	Compilar un método	Eliminar un método
Creación de clase	×	×		×	×	×	×	×
Redefinición de clase		×						
Cambio de nombre								
Cambio de comentario				×				
Comentar variable					★			
Eliminar una clase	×	×		×	×	×	×	×
Compilar un método							✷	✷
Eliminar un método							✷	✷

× En caso de tratarse de la misma clase

★ En caso de tratarse de la misma variable y clase

✷ En caso de tratarse del mismo método y clase

Tabla 8-2

Generar y mantener toda la información necesaria para deshacer los efectos del uso del sistema provoca en algún momento sobrecarga a nuestro ambiente. Sin embargo, analizando las definiciones de las operaciones vemos que en ciertos casos se pueden obviar datos sin incurrir en perdidas de información. Por ejemplo, considere el caso donde la clase Example ha sido definida. Si nos encontramos luego ante la modificación de su comentario podemos obviar este hecho y evitar que

recordar el comentario anterior para luego restablecerlo. Esto se debe a que el mecanismo que se aplicará para deshacer la creación de la clase no depende del comentario y a su vez deshace también su cambio.

Esta particularidad de inclusión entre acciones se presenta en otros casos y puede ser aprovechada para simplificar la información de restauración que es necesario mantener. La Tabla 8-2 indica aquellos eventos (en columnas) que no necesitan ser recordados en caso de presentarse algún otro en particular (el filas).

Si bien el total de elementos a almacenar crecerá con el tiempo es de esperar que no supere ciertos limites. La mayor parte de las actividades de los alumnos son localizadas en grupos de clases y métodos y de observar lo expuesto se deduce que en estas condiciones la necesidad de almacenamiento es menor.

Debemos considerar ahora una estrategia para detectar la ocurrencias de estos eventos determinar. Las relaciones mostradas en la Tabla 8-2 serán de utilidad para decidir si un cambio debe ser recordado o no.

8.1.2 Detección de Cambios

Como se menciona con anterioridad los cambios se originan en el trabajo con las herramientas de programación y el la carga de libros.

Las actividades de programación se desarrollan en las páginas de los libros de programación, que son parte integral del framework de programación LearningWorks. Por las características del mismo no sería difícil desarrollar nuevos modelos para cada herramienta con la capacidad de notificar cada vez que una nueva acción se produzca. Sin embargo, es de esperar que los autores de LearningBooks desarrollen sus propias páginas de programación. En este caso, el anterior enfoque vuelve muy complicada la tarea de los autores dado que los involucra en el sistema de detección y reversión de cambios.

La alternativa más prácticas es modelar un único punto de entrada a donde pueden requerirse todas las operaciones de extensión y modificación del modelo de objetos. Este sistema detecta trasparentemente la ocurrencia de los cambios al mismo tiempo que esconde el proceso y provee una interface de acceso clara. Esto es posible dado que el número de operaciones es limitado. De no ser así, la interface con este sistema crecería de forma indiscriminada atentando contra el diseño de objetos del mismo.

Este modelo también es viable en el contexto de la carga de libros. Si se mantiene suficiente información en un libro almacenado como para instruir a este sistema en la instalación de las clases y métodos se consigue un resultado análogo al de las herramientas de programación. A partir del momento en que los todos los cambios se integran en un único repositorio, el origen de los mismos deja de tener importancia.

De esta forma conseguimos un mecanismo que gracias a la recolección de cambios al ambiente permite su reversión, asegurando que luego de la descarga del libro el ambiente se encuentra en su estado original (anterior a la carga).

8.2 Extensión del Modelo de Visión

Limitar y configurar el acceso a las operaciones que pueden efectuar los usuarios impacta también en la integridad del ambiente. Cierto tipo de operaciones sobre algunos de los objetos del corazón de Smalltalk puede llevar el ambiente a dejar de funcionar. Por ejemplo, un cambio experimental en la forma en la que los objetos Rectángulo se comparan y todas las ventanas del ambiente dejan automáticamente de funcionar.

Esta apertura es uno de los principios de la filosofía Smalltalk. Todo esta disponible y puede ser cambiado. Esta característica bien puede sonar como un defecto pero es en realidad su mayor virtud. En ambiente es tan abierto que permite por ejemplo la aparición de LearningWorks o la introducción de interpretes de lenguajes lógicos. La existencia de un modelo de objetos para el lenguaje y ambiente permite que este evolucione de acuerdo a las necesidades del usuario; da lugar también a la construcción de aplicaciones que pueden ser retocadas mientras ejecutan o que se corrigen a si mismas. El modelo es tan rico que se vuelve en una fuente de inspiración para quienes lo habitan.

En un contexto de aprendizaje este aspecto de Smalltalk tiene lado debil. Un docente no puede esperar que un alumno sepa lo que hace cuando se involucra con las clases básicas del ambiente. En situaciones experimentales esto ser útil para demostrar las dependencias entre las partes de un sistema y la importancia de conocer los contratos a los que están sujetos los objetos en él. Sin embargo hay casos donde es importante proveer al alumno de una herramienta segura y estable donde explorar los conceptos del curso. En estas circunstancias, tener que pensar en las restricciones del modelo que soporta la herramienta es sobrecarga innecesaria.

La riqueza de la biblioteca de clases de Smalltalk, con mas de 1000 clases, también se vuelve un problema durante el aprendizaje. Por cada actividad que el alumno desarrolla se enfrenta con tantas alternativas que el solo evaluarlas insume más tiempo que el total previsto. Las herramientas y mecanismos necesarios para manejar tanta cantidad de recursos se convierten en barreras que el alumno debe atravesar para resolver cada simple ejercicio. Para saltear estos inconvenientes LearningWorks introduce el concepto de visión.

La Visión (presentada en el capítulo 5) es una propiedad de un libro o sección que indica que clases y métodos van a ser presentados al alumno por las herramientas de programación. Gracias a este mecanismo el autor del libro consigue enfocar la atención del alumno a los aspectos mas relevantes del modelo de objetos a explorar. Al mismo tiempo, dado que disminuye la cantidad de elementos con los que se debe trabajar, las herramientas de exploración se vuelven mas simples.

En el modelo implementado en la versión 0.8 de LearningWorks la visión se consigue combinando inclusiones y exclusiones. Las inclusiones declaran todos aquellos elementos que van a ser presentados a alumnos. Las exclusiones declaran aquellos de los incluidos que van a ser excluidos. La función de las exclusiones (que parecen innecesarias dado que en lugar de excluirlo se debió haber optado por no incluirlo) se hace necesaria cuando se utilizan comodines para simplificar la enumeración de las inclusiones. Por ejemplo, si al autor le interesa dar acceso a todas las clases de tortugas pero no a la clase CommanderTurtle puede enumerar todas las clases que quiere incluir en la propiedad include o simplemente indicar *Turtle* en el include y CommanderTurtle en el exclude.

Junto con la declaración de un clase en la cláusula include o exclude se indica que debe suceder con sus métodos. Las variantes van desde indicar que todos los métodos de la clase se incluyen o excluyen (utilizando un comodín en la declaración de los métodos en el include o exclude respectivamente) hasta indicar para cada uno de ellos el nivel con el que va a poder ser accedido. Esto es 0, indicando acceso completo de lectura y escritura al método y 2, indicando acceso solo de lectura al comentario y selector del método. La meta en este siguiente nivel de especificación de acceso es exponer a los alumnos ante situaciones de trabajo donde los principios del paradigma se vuelven más obvios e importantes. Por ejemplo, proveer a un alumno de una clase donde solo puede acceder a los comentarios y selectores de los métodos lo fuerza a trabajar en un contexto donde debe confiar en los protocolos declarados por los objetos desconociendo su implementación. Esta es una lección que no puede faltar en un curso introductorio de objetos.

Esta flexibilidad que se consigue en el manejo de métodos, gracias al detalle de nivel de acceso, no fue prevista para las clases. Sin embargo es común encontrar ejemplos donde es interesante

disponer de dicha funcionalidad. Por ejemplo podríamos evitar que un alumno tenga acceso a modificar una clase pero a su vez indicarle que debe hacer una extensión a la misma. De esta forma lo forzamos a explorar herencia como una forma de extender la funcionalidad. Este mecanismo de control de acceso a las clases también es de utilidad para evitar que el alumno tenga acceso a la representación interna de la misma (forzando el ocultamiento de información) y que introduzca cambios que pongan en peligro el funcionamiento del ambiente o micro-mundo.

8.2.1 Nuevo Modelo de Visión

Para conseguir un control de acceso flexible debemos considerar la información y operaciones mas importantes relacionadas a clases y métodos. Para las clases tenemos: ver su descripción pública (comentario y lista de métodos que implementa), explorar su implementación (ver la definición de su estructura interna), extender su implementación (modificar la estructura y agregar métodos) y eliminar (quitarla del ambiente por completo). Para los métodos tenemos: ver su descripción pública (selector y comentario), explorar su implementación (tener acceso al código fuente), extender su implementación (modificar el código fuente) y eliminarlo.

Los distintos niveles de trabajo y exploración descriptos sugieren un esquema de permiso de acceso a clases y métodos uniforme donde de forma incremental se da control al usuario hasta darle control total. El mismo se compone de cuatro niveles de acceso para el trabajo con clases y cuatro para el trabajo con métodos identificados como browse (ver), explore (explorar), extend (extender) y remove (eliminar). Cada uno de ellos extiende al anterior dando al usuario más atribuciones. La Tabla 8-3 describe este esquema en detalle.

Método		**Clase**
Ver el selector y comentario	browse	Ver el comentario
Ver el selector, comentario e implementación (código fuente)	explore	Ver el comentario y la definición de estructura.
Ver el selector. Ver y modificar el comentario y la implementación.	extend	Ver y modificar el comentario y la definición de estructura. Agregar nuevos métodos
Ver el selector. Ver y modificar el comentario y la implementación. Eliminarlo.	remove	Ver y modificar el comentario y la definición de estructura. Agregar nuevos métodos. Eliminarla.

Tabla 8-3

Los permisos son establecidos por el autor durante el desarrollo del libro. Para simplificar dicha tarea (que se vuelve compleja porque se debe indicar uno por uno los permisos de cada clase) tomaremos del modelo de visión original la idea de los comodines y la separación en inclusión y exclusión. Será posible entonces indicar utilizando comodines los permisos que se tienen para un dado conjuntos de elementos refinando, luego, al indicar cuales se quieren quitar. El esquema de definición de visión queda entonces compuesto por dos cláusulas: allow (permitir) e inhibit (limitar), cada una de ellas agrupando un conjunto de declaraciones de permisos de clase y método. La Ilustración 8-1 muestra un ejemplo de una declaración visión valiéndose del nuevo esquema..

```
vision Vision {
        allow [
                ClassAccess {
                        name '*Turtle*'
                        rights [ 'explore', 'remove', 'extend', 'browse']
                                instanceMethods [
                                        MethodAccess {
                                                selector '*'
                                                rights [ 'explore', 'remove', 'extend', 'browse']}]
                                classMethods [
                                        MethodAccess {
                                                selector '*'
                                                rights [ 'explore', 'remove', 'extend', 'browse']}]
                        }]
        inhibit [
                ClassAccess {
                        name 'CommanderTurtle'
                        rights 'browse'
                        }]}
```

Ilustración 8-1

Durante la construcción de libros el autor indica para algunos elementos cual es el nivel de acceso. Todos aquellos que no son mencionados no serán tenidos en cuenta por las herramientas de programación. Como resultado del trabajo del usuario irán apareciendo nuevas entidades (métodos y clases), para ellas se tendrá nivel de acceso total (remove).

El modelo de visión extendido de esta manera enriquece el ambiente al mismo tiempo que contribuye a mantener la integridad del mismo.

8.3 Integración

Los cambios necesarios para llevar a cabo las extensiones propuestas al modelo de visión y mecanismo de introducción afectarán principalmente a las clases del framework de programación. Por un lado, dada la nueva configuración de visión, las herramientas deberán extenderse de manera acorde

para hacer uso de las nuevas variantes. Por el lado del mecanismo de reversión de cambios, se deberá extender el mismo framework para asegurar que las herramientas de programación llevan cuenta de los cambios efectuados almacenándolos de forma segura.

Estas extensiones desencadenarán cambios, aunque de menor costo, el lenguaje de definición de libros (para modelar la nueva visión) y en el modelo de clases de visión. Igualmente se verá afectado el mecanismo de almacenamiento de libros en archivo dada la necesidad de incluir información que permita determinar los cambios originados por la carga de libros.

Debe tenerse en cuenta que como objetivos secundarios se busca que el efecto de estas extensiones sobre el proceso de desarrollo de libros y herramientas de programación sea mínimo.

8.4 Referencias

[1] Coplien, James O., Schmidt, Douglas C., eds. Pattern Languages of Program Design. Addison Wesley, 1995. ISBN: 0-201-60734-4

[2] Gamma, Erich, Helm, Richard, Johnson, Ralph and Vlissides, John. Design Patterns.Elements of Reusable Object-Oriented Software. Addison-Wesley Professional Computing Series, 1995. ISBN 0-201-63361-2.

[3] Goldberg, Adele, Abell, Steve, y Leibs, David. The LearningWorks Development and Delivery Framework, Communications of the ACM, October, 1997

[4] Goldberg, Adele, Robson, David. Smalltalk-80. The Language and its Implementation. Addison Wesley, 1983. ISBN 0-201-11371-6

[5] Goldberg, Adele. OOPSLA 95 Keynote, Addendum to the Proceedings of the ACM OOPSLA Conference, 1995.

[6] Riel, Arthur J. Object-Oriented Design Heuristics. Addison Wesley, 1996. ISBN 0-201-63385-X

Capítulo 9

Nuevo Framework de LearningWorks

Este capítulo presenta en términos generales las modificaciones de diseño necesarias para resolver las extensiones propuestas en el Capítulo 8. La sección 9.1 muestra como inspirados en los patrones de diseño Façade y Wrapper conseguimos soportar nuevas restricciones de visión. La sección 9.2 ilustra como el patrón de diseño Command sirve de inspiración para construir los mecanismos de soporte de reversión de cambios. Finalmente, la sección 9.3 presenta observaciones generales con respecto los cambios efectuados y a los resultados conseguidos.

9.1 Soporte al Nuevo Modelo de Visión

Las características de visión de un libro o sección deben ser interpretadas por las herramientas de programación para producir el efecto deseado. En el modelo original de LearningWorks los objetos utilizados para modelar la visión eran mayormente estructuras de datos y toda la responsabilidad de saber que hacer con ellas recaía en las propias herramientas de programación. Dadas la cantidad de variantes que introduce el nuevo modelo, esta arquitectura deja de ser suficiente. Es necesario idear nuevos objetos capaces de asistir a las herramientas de programación y a su vez permitir que el desarrollo de las mismas se simplifique.

Con el nuevo modelo de objetos aparece además de la necesidad de saber cuales clases mostrar y cuales no, y la responsabilidad de determinar que tipo de información mostrar en cada situación y que tipo de operaciones son factibles.

Los objetos involucrados hasta ahora son: los que modelan la visión, las herramientas de programación, las clases y los métodos. En los dos primeros no podemos delegar estas nuevas responsabilidades porque los volveríamos muy complejos. El problema con las clases y los métodos es que por estar estos completamente ligados al modelo del ambiente, cualquier cambios que hagamos directamente sobre ellos podría tener efectos secundarios indeseables. Debe encontrarse una

alternativa que permita extender estos objetos con el objetivo de delegar en ellos nuevas responsabilidades pero sin tener que modificarlos directamente.

El patrón de diseño Wrapper acerca una alternativa a este problema. Su objetivo es agregar responsabilidades adicionales a un objeto, tal vez de manera dinámica. Se presenta como una alternativa a la subclasificación como mecanismo para extender la funcionalidad. La Ilustración 9.1 muestra la estructura del patrón. La componente concreta define el componente al que se pretenden agregar responsabilidades. Componente define una interface común (no siempre existe en lenguajes no tipados como Smalltalk). Determina cual es la interface de todo decorador y mantiene una referencia a su componente. El decorador concreto agrega las responsabilidades.

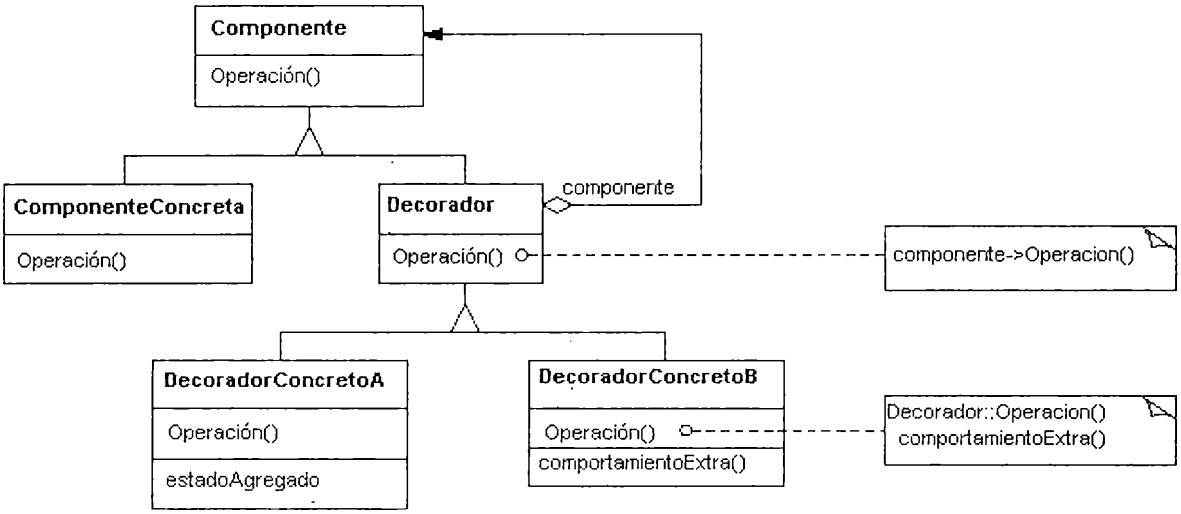


Ilustración 9.1

En nuestro caso las componente concreta serán clases y métodos. Toda operación a efectuar sobre ellos se dirige a sus decoradores. Sin embargo, nuestro caso no es una aplicación real del Wrapper. El elemento a decorar y su decorador no son intercambiables desde el punto de vista de quien los usa. Dado que los decoradores no solo extienden la funcionalidad de los elementos que decoran sino que también agregan nuevas responsabilidades referentes al control de permisos, se opto por denominarlos Chaperones (inspirados también en el patrón de diseño Bodyguard). Un chaperone extiende la funcionalidad del objeto al que representa y se convierte en la única vía de acceso al mismo. Por consiguiente, las herramientas de programación no trabajan directamente con clases sino con Chaperones de clases (ClassChaperone). No trabajan directamente con métodos sino con Chaperones de métodos (MethodChaperone). A continuación se describen brevemente sus responsabilidades principales.

ClassChaperone: Implementa métodos para cambiar la superclase de su clase; comentar variables de clase e instancia de su clase, cambiar el comentario de su clase, compilar nuevos métodos y agregarlos a su clase, definir nuevas variables de instancia y clase para su clase, eliminar su clase del sistema. También permite acceder a los nombres y comentarios de las variables de su clase, al comentario de su clase, a la definición de la clase y a la superclase de su clase.

MethodChaperone: Implementa métodos para eliminar el método al que representa, y para acceder a su selector y representación adecuada de su código fuente.

Los chaperones esconden todo el conocimiento y comportamiento relacionado al manejo de la visión que se refiere a ellos. Las herramientas solo delegan las operaciones y consultas en los chaperones y ellos respondes en relación a lo que conocen de la definición de visión.

Los chaperones se crean dinámicamente cada vez que una herramienta de programación los necesita. En el momento de su creación se configuran teniendo en cuenta los permisos definidos en la visión. Para desligar al programador de herramientas de la construcción y configuración de chaperones aparece un nuevo objeto, el contexto de programación, modelado en la clase NewLwProgrammingContext (en realidad existía en el modelo original de LearningWorks, como LwProgrammingContext, pero con menor responsabilidad). Este objeto es responsable de asistir a las herramientas en la construcción de chaperones.

En un principio el NewLwProgrammingContext implementa funcionalidad que hace posible conseguir todos aquellos chaperones de las clases con al menos permiso browse, el chaperone para una clase dada, los chaperones de las subclases de un chaperone. Los chaperones de métodos pueden pedirse por ahora al de la clase en cuestión. Cuando un explorador de clases necesita la lista de clases que debe mostrar al usuario, la pide al contexto de programación.

La complejidad de este nuevo esquema esta marcada por la necesidad del programador de herramientas de conocer la relación entre los chaperones y el contexto de programación y las operaciones individuales de cada uno de ellos. Cuando determinamos los objetivos de esta extensión nos propusimos que debía afectar lo menos posible al programador. Por consiguiente podríamos hacer que el contexto de programación actuara como ente concentrador de todas las operaciones. Según se discutió en el capitulo 8 esto es posible porque el numero de operaciones a efectuar es limitado y conocido.

El contexto de programación será ahora el principal punto de entrada de las herramientas de programación al sistema subyacente. Originalmente era encargado de conocer la clase con la que se

encontraba trabajando el usuario para así coordinar todas las herramientas. Ahora en lugar de conocer a la clase con la que se trabaja conocerá a su chaperone. Cada vez que se deba efectuar una operación, las herramientas se dirigirán al contexto de programación. Este a su vez cooperará con los chaperones dado que ellos son los que finalmente efectúan las operaciones y conocen las restricciones.

Un **NewLwProgrammingContext** debe implementar comportamiento que permita: conseguir todos los chaperones de las clases que se pueden mostrar al usuario, seleccionar uno de ellos como el foco de las herramientas (parte del diseño original de LearningWorks), conseguir los chaperones de los métodos de clase e instancia del chaperone seleccionado y que pueden mostrarse al usuario, aceptar un nuevo comentario para la clase del chaperone seleccionado, compilar un método para la clase del chaperone seleccionado, conseguir y establecer el comentario para una variable de la clase del chaperone seleccionado, definir una nueva clase y construir su chaperone, definir nuevas variables para la clase del chaperone seleccionado, eliminar un método o clase y conseguir el comentario de la clase del chaperone seleccionado. La Ilustración 9.2 esquematiza el diseño.

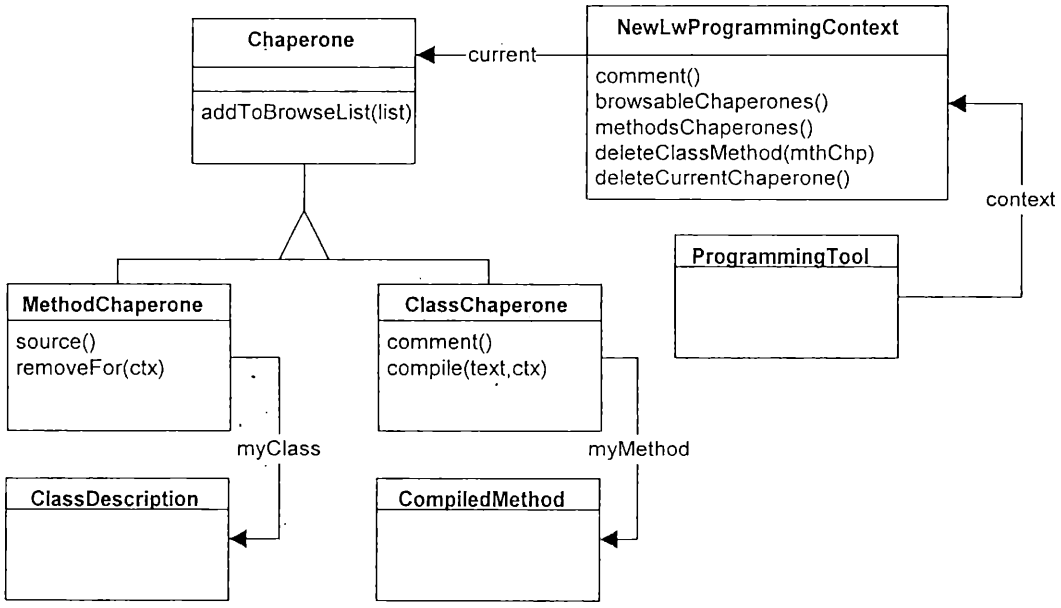


Ilustración 9.2

Esta arquitectura extiende el modelo de las herramientas de programación para hacer uso de las nuevas características de visión al mismo tiempo que esconde los cambios detrás de los chaperones. Futuras extensiones podrán introducirse con mayor simplicidad dado que solo debe instruirse a los chaperones al respecto.

9.2 Reversión de Cambios

Entre los aspectos discutidos en el Capítulo 8 con respecto al soporte de reversión de cambios se indica que para conseguir un manejo simple y extensible del mismo se debe contar con un único punto de entrada a las operaciones sobre el modelo de objetos. De esta forma se puede llevar control de toda operación efectuada y así mantener y actualizar la información de vuelta atrás. El patrón de diseño Façade aborda un problema similar a este. Su objetivo es modelar un punto de entrada único a un conjunto de subsistemas interactuantes. La Ilustración 9.3 muestra la estructura del patrón. El Façade conoce los subsistemas de los que es responsables y delega en ellos el trabajo. Los subsistemas son los que realmente implementan la funcionalidad.

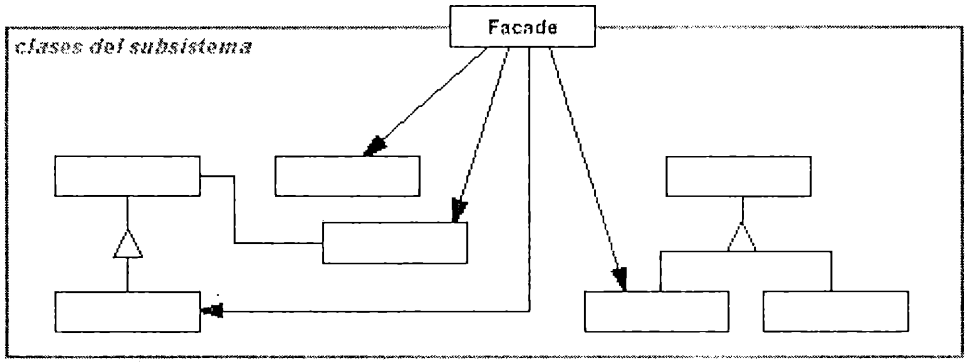


Ilustración 9.3

El contexto de programación, presentado en la sección anterior puede actuar como Façade, encerrando al sistema de programación y al de control y reversión de cambios. Por cada operación debe recabar y almacenar suficiente información como para luego permitir la reversión total de todas las operaciones.

El patrón de diseño Command tiene como uno de sus casos de aplicación aquel donde se necesita modelar las operaciones para luego deshacerlas. La Ilustración 9.4 muestra su estructura. El cliente es aquel que hace los requerimientos de la operación (en nuestro caso el contexto de programación). Conoce al sujeto de la operación (algún chaperone). Crea un comando de operación adecuado. El invocador es quien pide al comando que se ejecute (en nuestro caso también el contexto de programación).

Para nuestra aplicación contaremos con un tipo de comando por cada operación que queremos deshacer. agregar una clase, redefinir una clase, eliminar una clase, cambiar la superclase de una clase,

cambiar el comentario de una clase, cambiar el nombre de una clase, comentar una variable, compilar un método y eliminar un método. la ilustración 9.6 muestra la jerarquía de clases resultante.

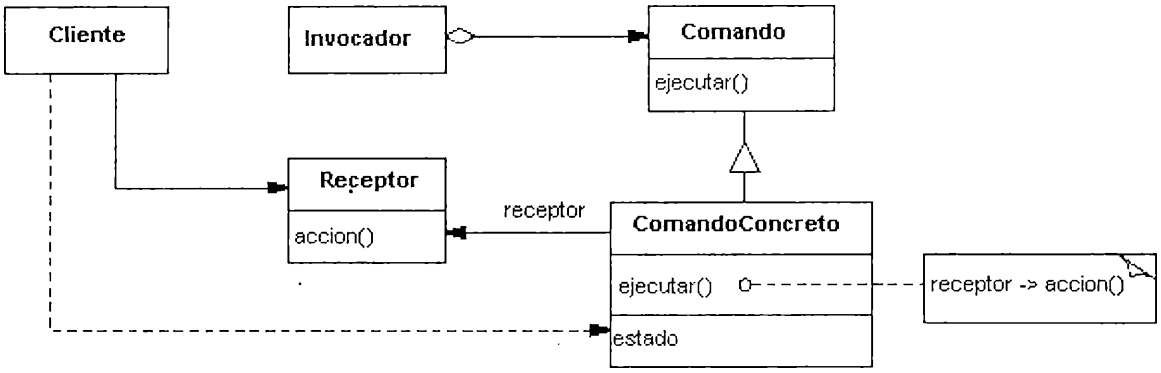


Ilustración 9.4

Cada vez que se le pide una nueva operación al contexto de programación, este en acuerdo con los chaperones involucrados instancia una de las clase de comando y luego de efectuarlo lo almacena. Luego de un tiempo de trabajo, el contexto de programación termina con una colección de comandos en su poder. De ser necesario recorrerá la colección empezando por el cambio más reciente pidiendo a cada uno que deshaga su efecto.

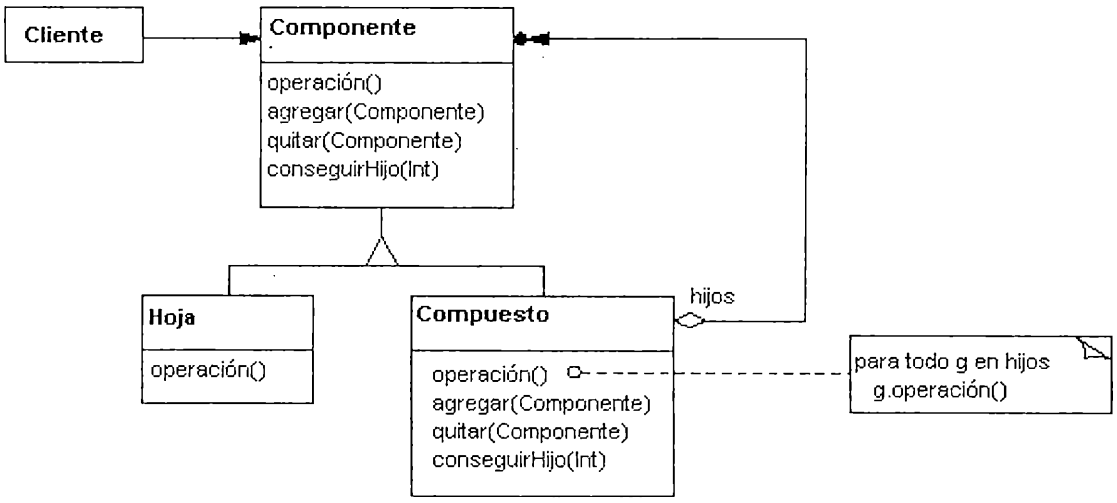


Ilustración 9.5

Cuando el capítulo anterior se propone el mecanismo guardar información de las operaciones se introduce una relación de inclusión entre ellas; el objetivo era descubrir en que casos se estaba frente a información irrelevante y actuar al respecto descartándola. En lugar de incluir esta

responsabilidad en el contexto de programación podemos observar lo propuesto por el patrón de diseño Composite. El objetivo del mismo es modelar objetos compuestos y simples para poder tratarlos de manera uniforme. Además delega en el objeto compuesto la responsabilidad de decidir como agregar sus componentes. El mecanismo compuesto por el patrón permite esconder en el comando compuesto la lógica de decidir que elementos almacenar y como. Cuando el compuesto recibe un pedido de deshacer su efecto, hace que todas sus componente se deshagan. La Ilustración 9.5 describe su estructura. Tanto el comando compuesto como el simple son capaces de responder al mensaje undo deshaciendo su efecto. La jerarquía resultante luego de modelas las operaciones como comandos siguiendo la estrategia del patrón Composite.

Para determinar si un comando debe formar parte del compuesto, este delega en cada una de sus componentes. Ellas interactúan con el nuevo comando para resolver si alguna de ellas subsume a la nueva caso en el cual no tendría sentido agregarlas.

UndoableCommand
- AddClassCommand
- ClassChangeCommand
  - RedefineClassCommand
  - RemoveClassCommand
  - SuperclassChangeCommand
- CommentClassCommand
- CommentVariableCommand
- CompileMethodCommand
- RemoveMethodCommand
- RenameClassCommand
- **ComposedCommand**

Ilustración 9.6

Combinando el Façade y el Composite conseguimos un modelo de manejo de operaciones simple de usar (dado que depende del protocolo implementado por el contexto de programación) y que a su vez es flexible y extensible (ya que al aparecer nuevas operaciones solo habrá que modelar nuevos comandos).

9.3 Las Herramientas de Programación

El paso final es reconstruir el framework de clases de programación para tener en cuenta este nuevo diseño. Para el autor de libros el único cambio será tener que trabajar con nuevas clases de interfaces, el esquema sigue siendo el mismo, una sección que es utilizada para construir libros de programación. Las páginas se obtienen a partir de esta como se explicó en la sección 5.4 sobre las herramientas de programación del framework. Para incluir una sección de programación en un libro

debe tomarse como interface de sección la clase NewLwBrowserInterface. Si bien las clases que modelan las distintas páginas de la secciones han sido reemplazadas por clases nuevas, esto es transparente para el autor. El mecanismo de resolución de interfaces y especificaciones oculta esto tras claves definidas por la sección. Los valores planteados en la Tabla 5.2 siguen siendo válidos.

Desde el punto de vista del constructor de herramientas de programación solo debe prestarse fundamental atención a los protocolos definidos por la clase NewLwProgrammingContext que modela el contexto de programación.

9.4 Referencias

[1] Beck, Kent. Smalltalk Best Practice Patterns. Prentice Hall, 1996. ISBN 0-134-76904-X

[2] Coplien, James O., Schmidt, Douglas C., eds. Pattern Languages of Program Design. Addison Wesley, 1995. ISBN: 0-201-60734-4

[3] Das Neves, Fernando y Garrido, Alejandra. Bodyguard: A Pattern for Object Distribution. Washington University Technical Report (#wucs-97-07)

[4] Gamma, Erich, Helm, Richard, Johnson, Ralph and Vlissides, John. Design Patterns.Elements of Reusable Object-Oriented Software. Addison-Wesley Professional Computing Series, 1995. ISBN 0-201-63361-2.

[5] Goldberg, Adele, Abell, Steve, y Leibs, David. The LearningWorks Development and Delivery Framework, Communications of the ACM, October, 1997

[6] Goldberg, Adele. OOPSLA 95 Keynote, Addendum to the Proceedings of the ACM OOPSLA Conference, 1995.

[7] Riel, Arthur J. Object-Oriented Design Heuristics. Addison Wesley, 1996. ISBN 0-201-63385-X

[8] Skublics, Suzanne, Klimas, Edward, Thomas, David. Smalltalk With Style. Prentice Hall, 1996. ISBN 0-13-165549-3

[9] Vlissides, John, Coplien, James O., and Kerth, Norman L eds. Pattern Languages of Program Design 2. Addison Wesley, 1996. ISBN 0-201-89527-7

Capítulo 10

Conclusiones

Este trabajo resume los resultados de nuestros esfuerzos en pos de mejorar el soporte que los ambientes de programación en objetos brindan en el contexto de aprendizaje de esta nueva tecnología. Si bien el caso de estudio y herramienta a la que se aplican las conclusiones de los estudios realizados es LearningWorks, estos son validos para todo otro ambiente de objetos. En particular si el aprendizaje se realiza de forma exploratoria y constructivista los aspectos de integridad, control de cambios y visión son fundamentales.

A lo largo de este documento se discuten los aspectos mas relevantes del aprendizaje de objetos y el rol que en este ámbito juegan los lenguajes de programación. Las opiniones y resultados aquí vertidos son el resultado de la investigación del autor en el área que no solo se limita al contexto de este trabajo de grado sino que forma parte de su carrera de investigación. Los dichos aquí expuestos han sido presentados y discutidos en diversas conferencias en el área de objetos y aprendizaje. Entre ellas podemos mencionar OOPSLA, ITiCSE, ECOOP y SCCC.

En lo que respecta a LearningWorks, los resultados de este trabajo serán considerados para inclusión en futuras versiones. De la discusión con algunos de los usuarios finales, como el caso de la gente de la Universidad Abierta del Reino Unido, se han obtenido comentarios muy favorables. En experimentos conducidos por el autor, se prueba que los cambios introducidos flexibilizan la herramienta al permitir preparar condiciones de exploración particulares para el alumno. Asimismo, aumentan la estabilidad del ambiente gracias a la reducción de la degradación progresiva producto de la carga y descarga de libros. Las nuevas combinaciones de control de visión también aportan en esta dirección.

Como se planteaba al enumerar los objetivos, se procedió a la migración de VisualWorks 2.5 a VisualWorks NC 3.0, este cambio no tuvo mayor impacto en los resultados obtenidos pero permitió la exploración de los nuevos mecanismos de empaquetamiento de código incluidos en la herramienta lo

que inspiro el nuevo modelo de empaquetamiento y reversión de cambios de LearningWorks. Durante el desarrollo de este trabajo se lanzo al mercado la versión 5i de VisualWorks que introduce el concepto de espacios de nombres; si bien este aspecto no es tratado en este documento, la implementación de los cambios asociados presenta avances con el objetivo de dar soporte a este nuevo mecanismo.

Agradecimientos

Este trabajo no hubiese sido posible sin el apoyo y la orientación de los directores Doctor Gustavo Rossi, Doctora Adele Golberg y David Leibs. Para ellos, mi sincero agradecimiento.

Agradezco especialmente el soporte del Laboratorio de Investigación y Formación en Informática Avanzada, LIFIA, y todos sus integrantes.

Capítulo 11

Referencias Bibliográficas

11.1 Libros

[1] Arnow, David M., Weiss, Gerald. Introduction to Programming Using Java : An Object-Oriented Approach. Addison-Wesley Pub Co, 1998. ISBN: 0201311844

[2] Beck, Kent. Smalltalk Best Practice Patterns. Prentice Hall, 1996. ISBN 0-134-76904-X

[3] Bergin, Joseph, Stehlik, Mark, Roberts, Jim, Pattis, Richard. Karel++: A Gentle Introduction to the Art of Object-Oriented Programming. John Wiley & Sons, 1996. ISBN: 0471138096.

[4] Coplien, James O., Schmidt, Douglas C., eds. Pattern Languages of Program Design. Addison Wesley, 1995. ISBN: 0-201-60734-4

[5] Eckel, Bruce. Thinking in C++. Prentice Hall Computer Books, 1995. ISBN: 0139177094.

[6] Eckel, Bruce. Thinking in Java. Prentice Hall Computer Books, 1997. ISBN: 0136597238.

[7] Flanagen, David. Java in a Nutshell : A Desktop Quick Reference (The Java Series). O'Reilly & Associates 2nd edition, 1997. ISBN: 156592262X

[8] Gamma, Erich, Helm, Richard, Johnson, Ralph and Vlissides, John. Design Patterns.Elements of Reusable Object-Oriented Software. Addison-Wesley Professional Computing Series, 1995. ISBN 0-201-63361-2.

[9] Goldberg, Adele, Robson, David. Smalltalk-80. The Language and its Implementation. Addison Wesley, 1983. ISBN 0-201-11371-6

[10] Goldberg, Adele; Rubin, Kenneth S. Succeeding with Objects. Addison Wesley, 1985. ISBN 0-201-62878-3

[11] Jacobson Ivar. Object-Oriented Software Engineering: A Use Case Driven Approach. Adisson Wesley, 1992. ISBN: 0-201-54435-0

[12] Kafay, Yasmin, Riesnik, Mitchel. Constructionism in Practice : Designing, Thinking, and Learning in a Digital World. Lawrence Erlbaum Assoc, 1996. ISBN: 0805819851

[13] Lalonde, Wilf R, Pugh, John R. Inside Smalltalk: Volume 1. Prentice Hall Engineering, Science & Math, 1990. ISBN 0-13-468414-1

[14] Lalonde, Wilf R, Pugh, John R. Inside Smalltalk: Volume 2. Prentice Hall Engineering, Science & Math, ISBN 0-13-465964-3

[15] Lalonde, Wilf R, Pugh, John R. Smalltalk/V: Practice and Experience. Prentice Hall Engineering, Science & Math, 1993. ISBN 0-13-814039-1

[16] Lalonde, Wilf. Discovering SmallTalk. Addison Wesley, 1994. ISBN 0-80-532720-7

[17] Lippman, Stanley B., Lajoie, Josee. C++ Primer. Addison-Wesley 3rd edition, 1998. ISBN: 0201824701

[18] Paper, Seymour A. Sculley, John. Mindstorms : Children, Computers, and Powerful Ideas. Basic Books (Sd), 2nd edition 1999. ISBN: 0465046746

[19] Resnik, Mitchel. Turtles, Termites, and Traffic Jams. Explorations in Massively Parallel Microworlds. MIT Press, 1994. ISBN 0-262-18162-2

[20] Riel, Arthur J. Object-Oriented Design Heuristics. Addison Wesley, 1996. ISBN 0-201-63385-X

[21] Rumbaugh, James, Blaha, Michael, Lorensen, Wiliam, Eddy, Frederick, Premerlani, William. Object-Oriented Modeling and Design. Prentice Hall Engineering, Science & Math, 1990. ISBN 0-13-629841-9

[22] Savitch, Walter, Szvitch, Walter, Johnsonbaugh, Richard. Java: An Introduction to Computer Science and Programming. Prentice Hall, 1998. ISBN: 0132874261.

[23] Skublics, Suzanne, Klimas, Edward, Thomas, David. Smalltalk With Style. Prentice Hall, 1996. ISBN 0-13-165549-3

[24] Stroustrup, Bjarne. The C++ Programming Languaje. Adisson Wesley Publishing, 1992. ISBN: 0201889544

[25] Vlissides, John, Coplien, James O., and Kerth, Norman L eds. Pattern Languages of Program Design 2. Addison Wesley, 1996. ISBN 0-201-89527-7

[26] Wirfs-Brock, Rebecca, Wilkerson, Brian, Wiener, Lauren. Designing Object-Oriented Software. Prentice Hall Professional Technical Reference, 1990. ISBN 0-13-629825-7.

11.2 Papers y Artículos en Journals

[1] Adams, Joel C. Object-Centered Design: A Five-Phase Introduction to Object-Oriented Programming in CS1-2. Proceedings de SIGCSE'96 (78-82). ACM PRESS, 1996. ISSN 0097-8418

[2] Beck, Kent, Cunningham, Ward. A Laboratory for Teaching Object-Oriented Thinking. Proceedings de OOPSLA'89. ACM Press, 1989.

[3] Cross, James H., Phillips, Thomas. Succesfully Integrating Traditional and Object-Oriented Approaches with Ada 95. Proceedings de SIGCSE'96 (19-23). ACM PRESS, 1996. ISSN 0097-8418

[4] Culwind, Fintan. Object Imperatives!. Proceedings de SIGCSE'99 (31-36). ACM PRESS, 1999. ISSN 0097-8418

[5] Das Neves, Fernando y Garrido, Alejandra. Bodyguard: A Pattern for Object Distribution. Washington University Technical Report (#wucs-97-07)

[6] Fernández, Alejandro, Gustavo Rossi et. al. A Learning Environment to Improve Object Oriented Thinking. Publicado en "Educator's Symposium Notes", OOPSLA'98. ACM Press.

[7] Goldberg, Adele, Abell, Steve, y Leibs, David. The LearningWorks Development and Delivery Framework, Communications of the ACM, October, 1997

[8] Goldberg, Adele. OOPSLA 95 Keynote, Addendum to the Proceedings of the ACM OOPSLA Conference, 1995.

[9] Kölling, M. Koch, B y Rosenberg, J. Requirements for a First Year Object-Oriented Teaching Language. Proceedings de SIGCSE'95 (173-177). ACM PRESS, 1995. ISSN 0097-8418

[10] Kolling, Michael, Rosenberg, John. An Object-Oriented Program Development Environment for the First Programming Course. Proceedings de SIGCSE'96 (83-87). ACM PRESS, 1996. ISSN 0097-8418

[11] Parlante, Nick. Teaching with Object Oriented Libraries. Proceedings de SIGCSE'97 (140-144). ACM PRESS, 1997. ISSN 0097-8418