Introducción a la Programación Concurrente en un Seminario de Lenguajes

Ing. A.De Giusti[1]

Laboratorio de Investigación y Desarrollo en Informática[2]
Departamento de Informática - Facultad de Ciencias Exactas
Universidad Nacional de La Plata

Resumen

Se analiza la experiencia docente de incorporar los temas inciales de Programación Concurrente (recursos compartidos, multiprocesamiento, sincronización, exclusión mútua) en un curso de segundo año de la Licenciatura en Informática de la UNLP.

En particular se discute el contexto de conocimientos del alumno, la bibliografía para los temas teóricos y el soporte del lenguaje para el desarrollo de prácticas, que en este caso ha sido ADA.

Tres años de experiencia en el curso permiten realizar una serie de reflexiones sobre las bondades y defectos del enfoque, reconociendo la importancia creciente de la Programación concurrente y paralela dentro de una currícula actualizada de Informática.

Por último se mencionan una serie de "problemas tipo" que enmarcan el desarrollo conceptual del curso, la vinculación con asignaturas posteriores así como diversas experiencias con alumnos utilizando distintas arquitecturas de soporte.

[1] **Inv. Principal CONICET. Profesor Tit. Ded. Excl., Dpto. de Informática, Facultad de Cs. Exactas, UNLP.**
E-mail degiusti@ada.info.unlp.edu.ar
[2] **Calle 50 y 115 Primer Piso, (1900) La Plata, Argentina, Teléfono 54-21-227707**
E-mail lidi@ada.info.unlp.edu.ar

Motivación

Es clara la tendencia creciente al procesamiento distribuído y al multiprocesamiento en la Informática actual.

En este contexto es importante para el alumno de Informática avanzar en el conocimiento de los temas de base tales como los conceptos de multiprocesamiento, procesamiento paralelo y programación concurrente a fin de adaptarse al cambio tecnológico de nuestros días.

En particular en la UNLP la Licenciatura en Informática tiene un curso específico de Programación Concurrente en 4to. Año, lo que parecía algo tardío respecto de lo básico del concepto para otras asignaturas (Sistemas Operativos, Sistemas de Tiempo Real, Sistemas Operativos Distribuídos).

Pareció interesante adecuar los contenidos teóricos de un Seminario de Lenguajes (2do. Año) a fin de poder introducir algunos conceptos propios de Lenguajes (en realidad los alumnos tienen en 3er. Año la asignatura Conceptos de Lenguajes) y de Programación Concurrente.

Naturalmente una opción posible (adecuada ? óptima ?) resultó el lenguaje ADA.

Elección del Lenguaje ADA

La elección de ADA como lenguaje para el Seminario no resultó trivial.

Tradicionalmente en este curso se habían dado muy diferentes lenguajes (incluso ADA en 1990) y también un enfoque más abstracto hacia Lenguajes Formales y Autómatas (que se mantiene como opción), pero casi siempre con un criterio *informativo.*

Las objeciones más importantes han sido la complejidad del lenguaje, la escasa experiencia en su utilización y la dificultad de conseguir un compilador accesible para los alumnos. Por otra parte, el salto conceptual desde Pascal parecía muy importante.

Las ventajas (especialmente mencionadas por los Profesores de Conceptos de Lenguajes y de Programación Concurrente) eran las soluciones que aporta ADA (sin salir del estilo procedural de Pascal que los alumnos conocían) y la posibilidad de introducir la concurrencia *con un lenguaje de implementación concreto.* Por otra parte el encapsulamiento de datos y los conceptos de ocultamiento y protección se resuelven perfectamente con los packages de ADA, avanzando notablemente respecto de las dificultades de manejo de abstracción de datos en Pascal.

En definitiva, se resolvió que el costo de no disponer de un ambiente amigable de desarrollo y de tener que entrenar a los auxiliares docentes se compensaba con la riqueza expresiva del lenguaje y las facilidades para la especificación correcta de algoritmos concurrentes.

Ejes Temáticos del Curso

Podríamos definir los principales ejes temáticos del curso en los siguientes tres puntos:

1- Insistir sobre conceptos fundamentales de lenguajes de programación, en particular abstracción de datos.

Este objetivo es complementario de lo que el alumno ha aprendido en primer año, que llega hasta el concepto de Tipo Abstracto de Datos y las limitaciones para implementar verdaderos TADs en un lenguaje como Pascal.

2- Introducir el concepto de concurrencia en programación y los mecanismos para especificarla.

Este objetivo debe ser formativo para el alumno y permitirle una apreciación global del problema de multiprocesamiento y el significado de la concurrencia para el hardware y para el software, así como los requerimientos para un lenguaje orientado a programación concurrente.

3- Reforzar una metodología de análisis y resolución de problemas utilizando el lenguaje ADA.

Este objetivo tiende a plantear al alumno problemas de mayor generalidad que los resueltos en primer año, buscando reforzar el aprendizaje de una Metodología de análisis y diseño de software, que posteriormente debiera concretar en las asignaturas de Sistemas e Ingeniería de Software.

Introducción a la Programación Concurrente utilizando ADA

Considerando los conocimientos previos de los alumnos del curso (fundamentalmente un curso anual de Algorítmica y Programación utilizando Pascal en la práctica), fue conveniente estructurar una jerarquía creciente con los temas y aplicar dos criterios:

-- Abstraernos de los recursos del lenguaje en el análisis y especificación del problema.
-- Plantear *siempre* un problema que sirviera de *foco* y sobre el cual se trabajara en un análisis "a la Pascal" y con programación concurrente en un lenguaje adecuado.

La jerarquía creciente de temas de la que hablamos siguió (aproximadamente) los siguientes lineamientos y pautas:

1- Separar el concepto de *paralelismo* (asociándolo con el hardware) del concepto de *concurrencia* (relacionándolo con el algoritmo, con el soft). Introducir la noción de proceso o tarea.

2- Marcar rápidamente la evolución de las arquitecturas de hardware hacia el multiprocesamiento, yendo de SISD a MIMD en una rápida visión conceptual, sin entrar en detalles físicos.

3- Establecer las ideas de *cooperación y competencia* entre procesos concurrentes, asociándolas con *paralelismo y exclusión.* Explicar la necesidad de *sincronización* y su importancia dentro de la programación concurrente.

4- Definir *el problema de la concurrencia* desde el punto de vista de la arquitectura física, de los sistemas operativos y de los lenguajes de especificación y programación de algoritmos. Como consecuencia analizar *los requerimientos de la programación concurrente para un lenguaje de programación.*

5- Discutir conceptualmente los mecanismos de *memoria compartida y pasaje de mensajes,* tratando que el alumno comprenda sus diferencias conceptuales y por qué la tecnología lleva a potenciar las soluciones de intercambio de mensajes en sistemas distribuídos.

6- Asociar los requerimientos para los lenguajes de programación concurrente con los modelos de solución en diferentes lenguajes. Explicar el rendezvous extendido de ADA, sin excluir otras alternativas (por ejemplo los canales half duplex de OCCAM).

7- Trabajar en la generalización conceptual de los *casos tipo* que se explican en el punto siguiente, buscando que el alumno supere la noción de "solución particular" y comprenda *la clase de problemas* a que se refiere cada ejemplo en cuestión.

8- Apoyarse en un modelo conceptual de procesamiento distribuído (que resulta claro para el alumno) para explicar los problemas de scheduling sobre datos y/o recursos físicos distribuídos.

Evolución de los problemas planteados al alumno

Desde el punto de vista de la ejemplificación, trabajamos con un gradiente de problemas (los enunciados están disponibles en el LIDI para los interesados), que se puede resumir del siguiente modo :

1- *Análisis de un servidor simple con M funciones y con N clientes idénticos.*

El primer modelo de problemas es el de un administrador de recursos compartidos (por ejemplo un mailbox) en el cual hay funciones diferentes (por ejemplo Depositar y Retirar del Mailbox) condicionadas por restricciones y N clientes idénticos que se comunican con el servidor.

Esta clase de problemas es muy simple y ayuda a que el alumno comprenda los mecanismos de exclusión mútua y sincronización, sin mayores complicaciones.

Por otra parte se puede generalizar tanto el servidor como los clientes utilizando la declaración de *tipo TASK* que provee ADA.

2- Análisis de un servidor simple con M funciones y clientes diferentes y con diferente prioridad.

La segunda clase de problemas se refiere a un administrador de recursos compartidos (por ejemplo un manejador de base de datos) en el cual hay funciones diferentes (por ejemplo Consultar o Escribir) condicionadas por restricciones y N clientes divididos en clases (por ejemplo Lectores y Escritores) con diferente prioridad y restricciones, que se comunican con el servidor.

Esta clase de problemas resulta excelente para que el alumno comprenda el concepto de *política de scheduling* ya que los clientes pueden resultar priorizados de diferente modo, según el código que escribamos en el server. Es interesante discutir variantes que afectan la "justicia" (*fairness*) de la política de scheduling elegida.

Un aspecto muy interesante que permite trabajar la clase de problemas de Lectores-Escritores es el análisis temporal simulado del acceso a los recursos por cada cliente, según la política de scheduling elegida.

3- Evolución del servidor simple de M funciones y N clientes idénticos hacia modelos de comportamiento adaptivo y con memoria.

El tercer modelo de problemas es el de un clásico adquisidor de datos remotos (1 o más) que "sirve" a un cliente de procesamiento de información, pero que debe tener la inteligencia de conocer parte de la historia del procesamiento (por ejemplo si ha adquirido un valor de señal determinada conocido como *alarma* o si el reloj de tiempo real supera un límite determinado) para cambiar *dinámicamente* su comportamiento (es decir la respuesta al servicio).

Esta clase de problemas resulta algo más compleja para el alumno, porque lo obliga a analizar los mecanismos de vinculación con el mundo real (por ejemplo con un reloj o una alarma) y a utilizar ciertos recursos de ADA (por ejemplo el manejo de los timers) con los que no ha experimentado en Pascal.

De todos modos la "adaptatividad" de la respuesta de un server ya la había experimentado al discutir variantes de la política de scheduling en problemas clase 2, teniendo en cuenta el estado de las colas de espera de servicio.

4- K servidores para N clientes idénticos donde los K servidores se intercomunican.

Este nivel de problemas corresponde al clásico ejemplo de los Filósofos y las Pastas (DIJ) y sus variantes. Normalmente 2/3 servidores (el de los filósofos, el de los tenedores y podría ser el del acceso a las sillas) atienden pedidos de procesos clientes que no se ven entre sí (los filósofos) y deciden intercambiando mensajes "intra-servers" antes de responder.

Esta clase de problemas es muy rica en conceptos como deadlock, inanición, fairness, recursos R-compartidos (es decir entre 2 procesos, R procesos o entre todos los procesos) y permite discutir la idea de "eficiencia" en el acceso a los recursos compartidos enfrentada con la idea de "justicia" en la distribución de los mismos.

Por otra parte se puede crear una notable variedad de alternativas que acercan al alumno a los problemas de scheduling típicos de los sistemas operativos.

5- Modelos de problema donde un proceso es cliente y servidor.

A este nivel interesa que el alumno comprenda que el atributo *cliente o servidor* puede cambiar dinámicamente. Elegimos entonces la modelización de problemas reales (por ejemplo una simplificación de tres niveles de cajeros bancarios) para mostrar que determinados procesos Pi son clientes de otros procesos Pj y al mismo tiempo servidores de otros procesos Pk. Más aún, discutimos con el alumno que 2 procesos pueden mutuamente ser cliente-servidor, según el tipo de servicio en juego.

Este análisis le permite al alumno dimensionar exactamente la potencialidad de rendezvous extendido como mecanismo de sincroniación y diferenciarlo de soluciones que requieren modelar *un* servidor para *una clase* de servicios.

6- Servidores generalizados para diferente tipos de clientes y servicios.

El útlimo modelo de problemas es el de un administrador de recursos generales y de distinto tipo (por ejemplo un servidor de periféricos de diferente tipo) en el cual hay pedidos diferentes dentro de los cuales puede haber funciones diferentes (por ejemplo tomar una impresora). Por otra parte, los mismos clientes pueden pedir diferentes recursos con distinta prioridad.

Esta clase de problemas es compleja y sólo se trata de que el alumno comprenda que un lenguaje como ADA brinda los recursos para resolverlos, resultando muy adecuado para la especificación/programación de aplicaciones de sistemas operativos.

Herrramientas para la Práctica

Una de las limitaciones más importantes (especialmente cuando se intentó por primera vez el Seminario en 1990) había sido el tipo de práctica posible por los recursos y por los conocimientos previos de los alumnos.

Actualmente trabajamos con una versión de ADA para Pcs (Janus ADA) y como alternativas tenemos Gw ADA y 2 versiones de ADA bajo Unix que son de dominio público y están disponibles en InterNet.

Más allá de las limitaciones del ambiente (y la emulación del multiprocesamiento que requiere) tanto el Janus ADA como el Gw ADA han resultado suficientes para el aprendizaje de los alumnos. La migración a compiladores bajo Unix tiene la dificultad natural de que el alumno no ve esta clase de Sistemas Operativos hasta tercer año.

Nuestra experiencia es que es muy importante que el alumno haga práctica sobre máquina y *sobre todo* tiene una gran valor que *cualquiera* de los compiladores utilizados respete la sintaxis y semántica establecida para el lenguaje. Por ello ponemos a disposición de los alumnos la definición formal de ADA y tratamos de discutir algunas limitaciones que aparecen en los compiladores.

La utilización de estos compiladores relativamente "pobres" ha servido para separar en el alumno la noción de "ambiente de desarrollo" de la de compilador del lenguaje, nociones que (utilizando tradicionalmente los ambientes Borland de Pascal) tienen bastante mezcladas.

Líneas de trabajo actuales

Más allá del trabajo estrictamente "de cátedra" en el Dpto. de Informática de la UNLP existe una línea de investigación en Programación Concurrente y Paralela y desde el punto de vista docente estamos trabajando en dos líneas para 1997:

-- Incorporar un ambiente de programación visual de 1 robot para el Ingreso y 1er. año de la carrera, *que pueda extenderse a varios robots.*

-- Incorporar el tema de programación de aplicaciones concurrentes con Redes Neuronales, como asignatura optativa.

Conclusiones

El autor considera conveniente introducir los temas de concurrencia y paralelismo en una etapa temprana del aprendizaje por los alumnos de Informática.

El multiprocesamiento (y la especificación del comportamiento de cada procesador y su vinculación con los otros) es un concepto ***natural*** en la Informática de nuestros días y no le resulta más complicado al alumno que la pogramación visual, la orientación a objetos o el modelo por capas de un protocolo como TCP/IP.

La respuesta de los alumnos en estos dos años ha sido excelente y actualmente estamos experimentando tres compiladores de ADA para reemplazar el original para PC que se viene empleando en las prácticas. Parece natural a tender a un ambiente bajo Unix o Windows NT con un modelo de compilador tipo ADA95.

Conceptualmente podría afirmar que el alumno está perfectamente capacitado para comprender los conceptos de concurrencia y paralelismo desde los inicios de su carrera en Informática y que incluso resulta muy estimulante para los alumnos desarrollar cierta capacidad de resolución de problemas concurrentes cercanos al mundo real.

Agradecimientos

El autor agradece especialmente al Profesor Marcelo Naiouf y al Lic. Fernando Tinetti sus útiles sugerencias en la definición del curso motivo de este artículo.

Bibliografía

[Usd81] "The programming language ADA - Reference Manual"
United States Department of Defense - Springer Verlag 1981

[And91] "Concurrent Programming".
Andrews. Benjamin/Cummings 1991.

[Hab83] "ADA for experienced programmers".
Habermann y Perry. Addison Wesley 1983.

[Set94] "Lenguajes de Programación. Conceptos y Constructores".
Sethi. Addison-Wesley 1994.

[Ghe91] "Programming languajes concepts".
Ghezzi, Jozayeri. Wiley 1991.

[Nie92] "ADA in distributed real time systems".
Nielsen. Mc Graw Hill 1992.

[Deg90] "Abstract machines in a first course of Computer Science. A critical analysis"
De Giusti, Lanzarini, Madoz. Proceedings of the 11th International Symposium "Computer at the University". Zagreb 1990.

[Cha96] "Herramienta visual para la enseñanza de programación estructurada"
Champredonde ,De Giusti. Enviado a la 2 CACIC.