ADA:
LENGUAJE DE COMPUTADORES
DE SISTEMAS DE DEFENSA

Juan L Hernández S.
Ph. D.

La Máquina Analítica entrelaza
patrones algebraicos justo como
el telar Jacquard teje flores y hojas.

Ada Byron

Introducción

Ada[1] es un moderno y poderoso lenguaje de programación de computadores digitales, que ha sido especificado y auspiciado por el Departamento de Defensa de Estados Unidos para todo nuevo equipo o sistema de defensa, o armamentos, con computadores incorporados. El Departamento de Defensa (DOD), más aún, exigirá el uso obligatorio, y excluyente, de este nuevo lenguaje en tales sistemas.

El lenguaje fue desarrollado vía un concertado y considerable esfuerzo del DOD, con los mejores especialistas en software del mundo y con los centros universitarios más avanzados en dicha especialidad[2]. Las especificaciones definitivas, a cumplir por el lenguaje, fueron establecidas en forma estricta como culminación de una secuencia evolutiva de documentos, en los cuales participaron grupos de especialistas del mundo, en particular de Estados Unidos y de algunos países europeos.

La versión definitiva del lenguaje fue llamada Ada por el DOD, en homenaje al primer programador (a) que existió: Ada Byron (1815-1852) condesa de Lovelace e hija del poeta Lord Byron, una matemática que trabajó con Charles Babbage, el creador de las Máquinas de Diferencia y Analítica, precursoras del computador digital moderno; ella programó la Máquina Analítica. Los estrictos requerimientos del lenguaje están prescritos en el documento citado al final de este artículo como referencia (1) y en la norma militar ANSI/DOD MIL-STD-1815, número este último en recuerdo del año de nacimiento de Ada Byron. Es costumbre que las publicaciones del DOD y de otros autores, sobre Ada, lleven la efigie de dicha dama, la que pictóricamente identifica el lenguaje.

Ada ha sido diseñado empleando los métodos y técnicas más avanzados y exitosos de la ingeniería de *software*, para cumplir diversas necesidades y objetivos de los sistemas de defensa con computadores incorporados. Por computador incorporado (*embedded computer*) se entiende uno que forma parte integral, del sistema o equipo respectivo. En sistemas de gran tamaño, como, por ejemplo, C3 (comando-control-comunicaciones) o detección temprana y otros, se requiere macrocomputadores o redes de grandes

---

[1] Ada (no es acróstico) es marca registrada del Departamento de Defensa de Estados Unidos (Ada) Joint Program Office).

[2] En este artículo se usará, por brevedad y por ser costumbre general, los vocablos *software* y *hardware*, ampliamente conocidos. Las traducciones españolas no son de aceptación general.

computadores. En armamentos o equipos, individuales o integrados se requiere desde minicomputadores hasta microcomputadores o microprocesadores, formando un componente o pieza más, pero con cierta inteligencia programable.

Dada la obligatoriedad impuesta por el DOD sobre el uso de Ada en todo equipo o sistema militar nuevo, hay una enorme actividad de las firmas de computadores y de *software*, tanto antiguas como otras de reciente formación, para desarrollar sistemas que satisfagan las exigencias prescritas. Por militar se entiende aquí los tres servicios (Ejército, Armada y Fuerza Aérea) que participan en la Oficina del Programa Ada (AJPO).

Huelga mencionar que los contratos del DOD envuelven sumas considerables, y que aquellas firmas o empresas no idóneas en Ada quedarán fuera de competencia en los actuales y futuros proyectos del DOD, prácticamente en todos los cuales hay computadores incorporados. Por otra parte, dadas sus características de diseño basado en lo más moderno de la ingeniería de *software* —y con diversas innovaciones originales, no conocidas antes— Ada presenta grandes ventajas para uso en sistemas "civiles", tales como los de control automático, comunicaciones, procesamiento de datos y muchos otros.

Ada es un lenguaje de alto nivel considerado como "el lenguaje de la década de los años 80", con considerables ventajas sobre otros existentes, tales como Fortran, Cobol, PL/I Algol, Pascal, y otros.

Por estas razones será inevitable la presencia de Ada en los años venideros, tanto en el ámbito civil como militar, aun en los países en desarrollo. El autor espera que este artículo sea útil para despertar interés en este poderoso y moderno lenguaje de computación. El enfoque es descriptivo, dado el carácter de la revista, y no se entra en detalles técnicos del lenguaje. En la referencia (2) aparecen artículos que describen la historia y detalles del desarrollo del “lenguaje común del DOD", que culminó en Ada. En (3) se puede apreciar la modularidad del lenguaje. Los textos (3-5) son apropiados para el estudio detallado de Ada. Pero, además del lenguaje mismo, ha habido, y hay, considerable esfuerzo en el desarrollo de compiladores, sistemas operativos y ambiente Ada, y las referencias (7-9) son útiles para una primera lectura de tales temas." Hay una voluminosa literatura sobre Ada, y su tasa de crecimiento es muy elevada.

El creciente costo del "software"

Desde la década de los años 50 se viene desarrollando una progresiva y creciente introducción de los computadores digitales en los equipos y sistemas de las Fuerzas Armadas de las diversas potencias. Por razones obvias, este artículo se referirá a los organismos estadounidenses, promotores de Ada, cuyo DOD ha impulsado muchos importantes avances en computadores y en sus aplicaciones, como es bien sabido, los que también han trascendido a empleo civil.

Las aplicaciones más conocidas de los computadores son las administrativas y de procesamiento de datos. Pero en las Fuerzas Armadas hay actualmente una amplia gama de aplicaciones de dichas máquinas, la que va desde grandes redes de computadores para la defensa de un continente, o detección de eventos en un ámbito mundial, hasta microprocesadores o microcomputadores incorporados en armamentos, vehículos, misiles, naves o equipos individuales. Estos computadores incorporados forman parte integral del equipo y deben trabajar en tiempo real, esto es, a la velocidad normal de los procesos de éste.

La tendencia a incorporar computadores en equipos y armamentos se ha acrecentado rápida y sorprendentemente desde la creación de la tecnología de los circuitos integrados y de *hardware* (componentes físicos), de bajo costo y pequeño tamaño. Estos últimos siguen decreciendo, lo que permite aumentar la incorporación de más "inteligencia" (*hardware*, más *software*) en los citados equipos. Aun más, hay en desarrollo tecnologías de circuitos integrados en escala muy alta y ultraalta, y el DOD está promoviendo también una tecnología secreta o reservada, de circuitos integrados de muy alta velocidad para uso en equipos militares, todo lo cual acelerará la mencionada tendencia a incorporar computadores.

Un aspecto interesante del problema de la incorporación de computadores es la no uniformidad de estos, lo que también aparece en otros ámbitos. En Estados Unidos hay una gran cantidad de fabricantes de computadores y equipos electrónicos digitales, y otros componentes diversos, los que compiten por los contratos del DOD, junto con empresas extranjeras, dada la economía "abierta" de esa superpotencia. Esto significa que en los sistemas militares se emplea una gran variedad de computadores diferentes entre sí, aun los de un mismo fabricante. Esta falta de uniformidad del equipo no necesariamente es grave, a nivel de usuario, puesto que en *hardware* los equipos computacionales deben cumplir normas estadounidenses o internacionales y, más aun, severas especificaciones militares. (MIL-STD) pertinentes. A niveles más internos, la falta de uniformidad de *hardware* causa problemas, que no interesan para el propósito de este artículo, dedicado a *software*.

Pero sí es grave el problema de la programación de esa gran cantidad de computadores diferentes. Cada fabricante de computadores, o de equipos computarizados, elige libremente uno o más lenguajes de programación para el caso. Cada computador puede ser programado en su lenguaje natural, o de máquina, pero estos son diferentes entre las diversas máquinas. Lo mismo ocurre con la microprogramación en el caso de computadores microprogramables. El uso de lenguajes ensambladores facilita la programación de un computador dado, pero no soluciona la desunifomidad citada.

Los lenguajes de alto nivel fueron desarrollados con diversos propósitos, siendo uno de ellos el que proveyeran una cierta facilidad de programación que fuera independiente del tipo de computador. Esto no se ha logrado completamente, en general. Además, estos lenguajes tienen varias versiones, generaciones y dialectos, o materializaciones diferentes, por los muchos fabricantes y para diversos computadores. En equipos militares hay, finalmente, lenguajes de alto nivel que no son de uso común de los programadores generales (Jovial, Tacpol, y otros).

Así, considerando que los costos de *software* van creciendo rápidamente a medida que el *hardware*, cada vez más poderoso, decrece en costo —y se automatizan cada vez más tareas— el DOD realizó en 1973 un estudio sobre el problema. Se dedujo que cerca de un 56 por ciento de los gastos de *software* anuales (que totalizaban unos 3 a 4 mil millones de dólares de esa época) provenían de los computadores incorporados. Se empleaba unos 200 modelos distintos de computadores y unos 450 diferentes lenguajes en tales sistemas. Los programas para computadores incorporados son muy largos —de 50 mil a 100 mil líneas— y de vida promedio de 10 a 15 años. Estos datos muestran que tales sistemas son muy diferentes de las que aparecen en centros de procesamiento de datos. El DOD estimó que al usar un lenguaje de programación común y único en sus computadores incorporados podría economizar, en el periodo 1983-2000, alrededor de 24 mil millones de dólares, una suma comparable a la deuda externa de algunos países del Tercer Mundo.

Cabe mencionar que el problema de la programación y su costo tiene varias facetas. Hay, primero, un costo de programación inicial de los computadores para el problema o

DOÑA ADA BYRON (1815-1852)

equipo especifico. Este gasto es inevitable, pero puede ser reducido empleando un lenguaje de alto nivel, estructurado, modular, con paralelismo, que sea muy conocido por los programadores involucrados y que tenga fuerte apoyo y soporte para programación. Estas son facilidades de edición, compilación, depuración, sistemas de desarrollo, y otros.

Otro aspecto importante es la mantención, modificación y mejoramiento de tales programas, de tan larga vida, y su adaptación a nuevos tipos de computadores o exigencias. Aunque el programador que debe modificar el programa sea el mismo que lo originó, muy rara vez recordará los criterios y trucos que empleó la primera vez y, generalmente, efectuará toda la programación de nuevo (decenas de miles de líneas de programa); a veces esto es agravado por el hecho de que él sólo escribió una parte del programa primitivo. También puede haberse usado primitivamente un dialecto ya en desuso al cabo de dos años. Dada la gran rotación de profesionales en Estados Unidos, es común, además, que la revisión o mejoramiento del programa deba ser hecha por otro programador, que ni siquiera conoce el lenguaje que se usó inicialmente (par ejemplo, un experto en Pascal que no sabe Lisp); lo más probable será que escribirá de nuevo todo el programa, no sólo la parte a modificar. Problemas como estos contribuyen inútilmente al oneroso gasto de *software* del DOD, o de cualquier empresa.

Para lenguajes generales de alto nivel hay muy buen conocimiento entre los programadores y, además, muchos soportes para aliviar y depurar la programación. Pero el DOD constató, por ejemplo, que Fortran y Cobol —dos lenguajes muy conocidos y generales— sólo representaban en conjunto como el 20 por ciento del *software* total. Los programadores gastaban mucho tiempo trabajando con lenguajes extraños, y, además, los programas no eran generalmente eficientes y confiables.

En sistemas militares complejos —como radares de advertencia temprana o guía de misiles crucero— es muy importante la confiabilidad y exactitud de los programas. Sin embargo, se advertían muchos errores y fallas de los programas, lo que se explicaba por las razones antes citadas. Por ejemplo, se encontró que en la vida de un programa típico participaba, en promedio, una decena de programadores diferentes, para mantenerlo, modificarlo, mejorarlo o adaptarlo a un computador nuevo o de marca distinta.

Evolución del lenguaje Ada, hasta el presente

Al advertir las deficiencias anotadas, y otras no indicadas, el DOD decidió implantar un lenguaje común para programación en computadores incorporados, es decir, para aplicaciones en problemas reales, a diferencia de otros lenguajes destinados a aprovechar mejor los computadores en procesamiento de datos y en otros géneros de usos. Sin entrar en muchos detalles de años anteriores, en 1975 se estableció un Grupo de Trabajo del Lenguaje de Alto Nivel (HOLWG, sigla en inglés), el que, con el aporte de destacados especialistas mundiales, produjo —entre ese año y 1979— una serie de documentos con especificaciones, cada vez más precisas y refinadas, sobre los requerimientos que debía cumplir el nuevo lenguaje. En 1979, el documento llamado con el nombre clave de Steelman, estableció los requerimientos definitivos del nuevo lenguaje.

Por brevedad, se puede resumir que el lenguaje debía ser apropiado para: *software* en computadores incorporados; diseño, desarrollo y mantenimiento de *software* confiable para sistemas grandes, de larga vida y permanentemente cambiantes; servir como lenguaje común (unificado), con posibilidad de ser estandarizado o normado en forma completa, sin ambigüedades e independiente de la máquina. Además, el lenguaje debía ser representativo

de las mejores prácticas y técnicas de la moderna ingeniería de *software*; proveer una base en torno a la cual construir un ámbito de desarrollo, mantenimiento y soporte de *software*; no imponer costos adicionales de ejecución en aquellas aplicaciones en que el lenguaje provea características inherentes que no sean necesarias o usadas.

Más adelante se considerará en detalle varias de las exigencias anotadas. Aquí explicaremos sólo la última exigencia. La idea del DOD y del HOLWG era, y es, disponer de un lenguaje único y rígido, no susceptible (o permitido) de ser parcelado en dialectos, subconjuntos o versiones, ni de ser extendido, puesto que de otro modo se volvería a la larga al problema (torre de Babel) de una profusión de lenguajes diferentes en el fondo o forma. (De hecho ya han aparecido muchas firmas de *software* con subconjuntos de Ada). Así, el lenguaje debería ser lo bastante complejo y poderoso como para poder programar en él los problemas más complicados y grandes en forma eficiente, en los sentidos ya citados. Por ello, en problemas promedios, el lenguaje estaría sobredimensionado y muchas de sus facilidades no serían usadas o necesarias. La sexta exigencia implicaría que al ejecutar estos programas, las características no necesarias o usadas no debieran ocupar memoria o tiempo adicionales del computador.

En el intertanto, el DOD había especificado que en cualquier nuevo *software* de defensa sólo se usara un lenguaje de alto nivel que estuviera dentro de una lista de siete, aprobados, entre los cuales estaban Ansi Fortran, Ansi Cobol y dos versiones de Jovial (Usada principalmente por la Fuerza Aérea).

En 1976 se contrastó unas dos docenas de lenguajes existentes, contra los requisitos elaborados hasta entonces (Steelman). En resumen, se encontró que ninguno de los lenguajes —incluyendo los siete interinos antes citados— servía como lenguaje común, pero que algunos derivados de Pascal, PL/I o Algol-68 podrían ser modificados para crear el lenguaje deseado.

De acuerdo con lo expuesto, se decidió producir un lenguaje común que cumpliera con todos los requisitos de Steelman y que fuera un derivado de Pascal, PI/I o Algol-68, aunque no necesariamente compatible con estos. Se llamó a concurso internacional sobre la base de algunos diseños preplaneados y se seleccionó cuatro proposiciones, de entre quince recibidas. Las firmas seleccionadas, cuyas propuestas estaban basadas en Pascal, comenzaron paralelamente sus diseños del lenguaje en 1977.

En 1979, con las conclusiones de varias decenas de evaluadores, se eligió el lenguaje propuesto por la firma Cll-Hoheyweil-Bull, bajo la dirección del experto francés Mr. Jean Ichbiah, en cuyo grupo había especialistas franceses, estadounidenses, ingleses y alemanes. Este lenguaje fue sometido a refinamientos y pruebas con problemas reales, pequeños (de no más de 5.000 líneas). Este lenguaje fue llamado Ada, por las razones antes expuestas.

En 1980 se completó la especificación final, definitiva y rígida y del lenguaje. Cualquier cambio ulterior, debe ser aprobado por el DOD (AJPO), que registró el lenguaje y su nombre. El lenguaje fue estandarizado por Ansi (una institución normalizadora) en 1983.

Además del lenguaje mismo, son importantes los aspectos de compilación y del ámbito de trabajo Ada, los que veremos superficialmente en las secciones que siguen.

Algunos conceptos sobre el lenguaje Ada

De lo dicho hasta ahora se infiere que Ada fue desarrollado para satisfacer primordialmente los requerimientos de computadores incorporados y también con base en

las mejores técnicas de la ingeniería de *software.* Además de lo dicho para sistemas con computadores incorporados, conviene agregar que se requiere o exige: control de entrada-salida de características especiales; procesamiento paralelo; control en tiempo real; manejo de excepciones, que alteran la secuencia natural de ejecución.

Los principios más relevantes de la ingeniería de *software* considerados en el diseño de Ada fueron: abstracción; enmascaramiento de información; modularidad; localización (contigüidad o proximidad física de partes de programas); uniformidad y competitividad; confirmabilidad. Una completa explicación sobre estos conceptos y la forma en que Ada los cumple está fuera del alcance de este artículo, pues demandaría muchas páginas. Sobre el lenguaje mismo, el lector debería consultar las referencias. Aquí solo se mencionará algunas características generales.

• Ada tiene una estructura de bloques que facilita una ilación lógica y ordenada de los programas. Las unidades programáticas que provee son: *subprograma*, que es una función o un procedimiento para expresar una sola acción: *paquete (package),* que es una colección de recursos computacionales que encapsula (encierra) tipos de datos, objetos de datos, subprogramas, otros paquetes o "tareas"; *tarea*, que es una acción ejecutada lógicamente (no necesariamente en sentido físico) en paralelo (concurrentemente) con otras acciones o tareas. Un sistema Ada puede consistir en cientos o miles de unidades programáticos. Cada unidad programática tiene, generalmente, una estructura de dos partes: la *especificación*, que contiene información visible para el usuario de esa unidad; el *cuerpo*, que contiene los detalles de su realización ocultos (enmascarados) para dicho usuario o interfaz.

• Ada es apropiado para metodologías de diseño diversas. Para metodología descendente (*top-down*) provee capacidad de compilación separada de subprogramas, paquetes y tareas, además de la facilidad de compilación separada de las partes de especificación y cuerpo de estas unidades programáticas. La metodología ascendente (bottom-up) es apoyada por Ada mediante la provisión de bibliotecas de paquetes. Estas y otras metodologías hacen que Ada sea muy útil también para el diseño de sistemas de *software* en general, no sólo para computadores incorporados.

• Ada provee modularidad declarativa u "orientada al objeto". Esto es logrado con las unidades programáticas citadas, especialmente con el "paquete". Las referencias a un paquete son a su parte de especificación; el cuerpo queda enmascarado. Las unidades programáticas son como subunidades compilables separadamente, como se dijo, y son mantenidas como objetos de base de datos en forma separada, pero sus partes de especificación quedan en los paquetes o subprogramas progenitores. Así, el programa ejecutable puede ser constituido al efectuar el proceso de enlace (*linking*).

• Ada, a través de los medios mencionados, provee o promueve confiabilidad, portabilidad, reusabilidad y mantenibilidad de *software*, que eran algunas de las metas al diseñar el lenguaje común (del DOD). Los módulos y unidades programáticas, y aun partes de éstas, pueden ser probados, compilados, corregidos, adaptados y mejorados individualmente.

• Ada es un lenguaje "fuertemente tipado", en el sentido de que cada dato debe tener un tipo. Un tipo de data (datos) define un conjunto de valores y un conjunto de operaciones aplicable a dichos valores. El compilador verifica la corrección de todas las operaciones de acuerdo al tipo de los datos, al compilar y al ejecutar los programas. Además de los tipos naturales que provee Ada (por ejemplo, enteros, reales, arreglos, récord, y otros); el usuario puede definir otros deseables. Por ejemplo, Ada permite tipos *privados*, usados sólo en paquetes, cuya estructura no es visible al usuario.

• Los tipos de data de Ada permiten la creación de estructuras de datos muy complejas, si se desea. Un ejemplo sería un *arreglo* dentro de un *récord*.

• Ada, dado que este principalmente diseñado para operaciones complejas en tiempo real (verbigracia, C3), soporta procecesamiento paralelo o concurrente. Provee concurrencia basada en un modelo de "tareas", que incluye ejecución paralela de estas, comunicación y cooperación entre ellas (por un mecanismo llamado *rendez-vous*, por ejemplo, original de este lenguaje) y protocolos de sincronización.

• Ada provee *procedimientos genéricos* y bibliotecas de ellos para aumentar la productividad de los programadores. Estos procedimientos (*procedure*) son programados solo una vez, con lógica independiente de los datos; éstos pueden ser especificados cuando se va a operar con ellos.

• Ada provee medios para manejar excepciones en tiempo real y, más aún, permite al usuario especificar en qué nivel desea manipular dichos casos. Así, Ada se puede recobrar, en tiempo real, de fallas de *hardware*, aritméticas, y otras. Los errores son corregidos al nivel lógico que impida su propagación.

Como un ejemplo muy simple de paquete Ada, en la figura 1 se muestra una parte de un programa ficticio sobre tipo y cantidad de buques de una fuerza naval. Las palabras subrayadas son términos reservados Ada. *Integer* es un tipo enumerativo definido en Ada. Los guiones entre palabras son para mejorar la legibilidad. Si se desea anotar comentarios aclaratorios, éstos se anteceden con un doble guión. El número cero se denota por ø, para evitar confusión con la letra O.

```
package  FUERZA-NAVAL  is
    --
    type  BUQUE-DE-GUERRA is (CRUCERO, FRAGATA, SUBMARINO);
    subtype CANTIDAD is INTEGER range Ø..1.ØØ;
    type TIPO-DE-BUQUE is
        record :
            NOMBRE: BUQUE-DE-GUERRA;
            NUMERO-DE-BUQUES. CANTIDAD;
        end record;
        --

      task NAVES-ACTIVAS is
      entry CAMBIOS (NAVES-ACTOVAS: out TIPO-DE-BUQUE);
      end NAVES-ACTIVAS;
  end FUERZA-NAVAL:
```

FIG. 1. EJEMPLO DE FRAGMENTODE PROGRAMA EN ADA

El compilador Ada

Un programa escrito en un lenguaje computacional de alto nivel debe ser traducido al lenguaje natural del computador, para que éste lo pueda ejecutar. En la práctica actual, la traducción la efectúa automáticamente el propio computador, mediante un programa llamado "compilador". Generalmente, la compilación se efectúa en uno o más pasos del programa original, llamado fuente por el computador. Hay un compilador especifico para cada lenguaje fuente, por ejemplo, Fortran, Pascal, y otros) y, en general, para cada tipo de computador y además la compilación es efectuada para el programa entero, no para partes

de él. En el caso de lenguajes "interactivos" se desea un diálogo entre el usuario y el computador; y se debe traducir cada instrucción separadamente. En este tipo de lenguajes, de los cuales es más conocido el Basic, el programa traductor es llamado interpretador o intérprete. El programa que resulte del proceso de compilación es llamado "programa objeto".

El proceso de compilaciones complicado y de acuerdo a la práctica moderna, se efectúa en varias etapas, bosquejadas en la figura 2.

En la primera etapa, o paso, el programa fuente es analizado lexicográficamente y dividido en unidades llamadas “fichas" (*tokens*) reconocibles, como palabras clave, identificadores, símbolos especiales, y otros. La salida es grabada en un archivo del almacenamiento (memoria) primario o secundario, según el caso. En el caso de Ada puede haber también instrucciones, llamadas "pragmas" al compilador las que son procesadas en esta etapa.

La retahíla de fichas del citado archivo es procesada luego por el analizador sintáctico (*parser*), que verifica la sintaxis y semántica y produce un archivo intermedio. Este código intermedio es procesado luego por el programa optimizador, que lee el código y trata de reducir operaciones redundantes, eliminar instrucciones innecesarias y aumentar la rapidez y eficiencia del programa resultante. El generador de código procesa el archivo optimizado y entrega un programa objeto reubicable. En cada una de las etapas o pasos indicados, el compilador verifica que no haya errores de ningún tipo, internos o de programa.

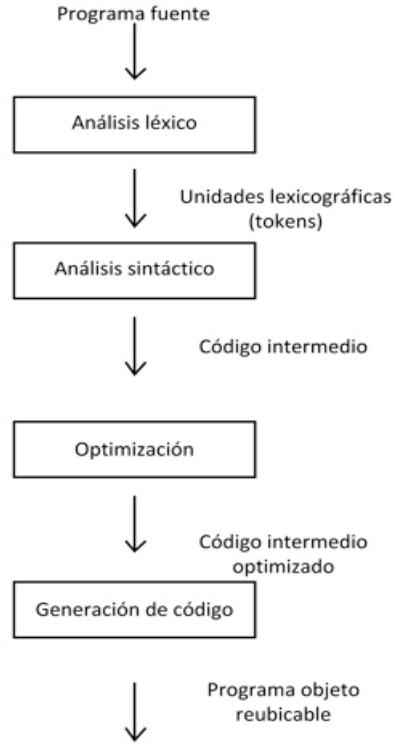

FIG. 2 ETAPAS DE UN COMPILADOR TIPICO

El archivo reubicable no es directamente ejecutable por el computador para entregar el resultado final buscado. Dicho código reubicable debe ser procesado por otro programa interno, llamado "cargador-enlazador", que entrega un código objeto ejecutable por el computador, como se ilustra en la figura 3.

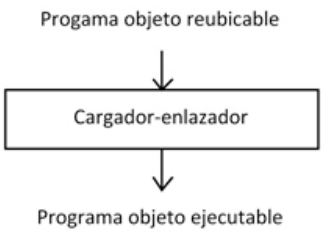

FIG. 3. OPERACION DE CARGA Y ENLACE

Como se explicó anteriormente, uno de los deseos del DOD era que el lenguaje común (Ada) proveyera portabilidad, mantenibilidad y actualizabilidad fáciles y económicas, además de las otras características óptimas de escritura y legibilidad. La portabilidad se refiere a la cualidad de un programa de ser ejecutable en una variedad de computadores distintos. Mantenibilidad quiere decir que el programa sea fácil de modificar, o alterar, a través de su larga vida (una o dos decenas de años) sin que haya que escribirlo enteramente de nuevo cada vez. Actualizabilidad implica, entre otras cosas, facilidad de aplicación a nuevos tipos de máquinas y situaciones. Hay muchos otros aspectos y detalles que no se discuten aquí, por brevedad.

Dadas las complejidades del lenguaje Ada, las exigencias adicionales recién citadas, lo difícil de la compilación y el deseo del DOD de evitar los problemas llamados "torre de Babel" (proliferación de diversas versiones y dialectos Ada) y "devorador monstruo de *software*" (insostenibles y crecientes costos de programación y programadores versados) el mencionado Departamento decidió rigidizar también las especificaciones a cumplir por los compiladores Ada en consonancia con la estrictez del lenguaje mismo.

A partir del documento llamado Stoneman (anterior al Steelman, antes citado en este artículo), el DOD decidió que los compiladores Ada y sus máquinas (computadores) deben ser capaces de ejecutar el lenguaje completo, no solo algún conjunto de éste, y encomendó a una firma de *software* (SofTech) la preparación de un riguroso conjunto de pruebas que un compilador Ada debe aprobar antes de ser validado por el DOD (AJPO). El conjunto inicial de validación contenía unas 1.200 pruebas, la tercera versión unas 2.000 pruebas, y se espera que lleguen pronto a unas 3.000 o más. Aunque la mayor parte de las pruebas son conocidas por los fabricantes, y públicas, y se agregan cada vez otras incógnitas, o reservadas.

Las validaciones duran un año, y al cumplirse ese período un compilador validado debe resometerse anualmente a las nuevas pruebas que se vayan estableciendo. El primer compilador Ada validado fue desarrollado por la firma Rolm, con computadores Data-General, en 1983. Dadas las exigencias de la validación, hasta los primeros meses de 1985 sólo se han validado unos cuatro compiladores, uno de los cuales (New York University) es meramente un intérprete para usos académicos. Un compilador Ada validado asegura que él y su computador pueden procesar el lenguaje Ada completo. Un compilador Ada que sólo pueda procesar un subconjunto del lenguaje no es válido para el DOD y para usos militares. Sin embargo, hay muchos fabricantes de computadores y firmas de *software* que ofrecen compiladores que pueden procesar los subconjuntos más útiles del lenguaje. Estos compiladores parciales son útiles para usos académicos o comerciales, en aplicaciones de menor envergadura que las que aparecen en sistemas de defensa con computadores incorporados. El autor ha usado un compilador Supersoft, para un subconjunto Ada, en un microcomputador Cromenco, ambos del Departamento de Electrónica de la Universidad Técnica Federico Santa María, de Valparaíso, Chile (Referencia 10).

Como se explicó en la sección anterior, una importante capacidad que provee Ada es la de poder compilar separadamente cada unidad programática y, más aún, las partes "especificación" y "cuerpo" de éstas. Así, el *software* se puede desarrollar, mejorar, probar o mantener en forma tan modular como se haría con el *hardware*, constituido por módulos de circuitos integrados u otros elementos. Un programa Ada es una interconexión de módulos de *software*, entrelazados por el compilador "tal como un telar teje flores y hojas".

El ambiente de programación y trabajo Ada

Además del lenguaje y los compiladores, la implantación del sistema Ada exige la creación de un Ambito de Soporte de Programación Ada (APSE, siglas en inglés), el que fue estandarizado en el documento Stoneman, ya citado. Un ámbito de programación es entendido como un dominio lógico y/o físico que contiene herramientas de *software*, lenguajes de programación, metodologías programáticas y programadores, destinado todo a desarrollar y materializar sistemas de *software*. El objetivo de APSE es proveer facilidades y medios para el desarrollo y mantenimiento de *software* de aplicaciones de Ada a través de su vida, período en que el programa es usado. Desde el punto de vista del DOD, se enfatiza el caso de aplicaciones con computadores incorporados.

APSE permite reducir costos de desarrollo de compiladores y otras herramientas de *software* y mejorar la portabilidad tanto del *software* como de los programadores mismos. Básicamente, adopta un enfoque de computador anfitrión (*host*) y computador destinatario (target), que pueden ser el mismo o muy diferentes, en cuyo caso el anfitrión es más poderoso que el destinatario. El APSE se aloja en el anfitrión, y en éste se desarrolla el *software* que usará el destinatario. APSE debe proveer una base de datos más amplia que la vista en el compilador, pues debe servir además para el manejo o gerencia del proyecto. Otro requisito de APSE es que pueda ser extendida y que todas sus herramientas sean escritas en lenguaje Ada. La firma Rolm, que creó el primer compilador Ada validado (1983), desarrolló una versión de APSE que llamó ADE, (Ada Development Environment).

Como es sabido, los recursos físicos y lógicos (o sea, de *hardware* y *software*) de un computador cualquiera son administrados, coordinadamente y en forma presumiblemente óptima, por un programa maestro llamado "sistema operativo". Un computador típico puede tener varios sistemas operativos (por ejemplo, DOS, UNIX, RTOS, y otros) pero sólo uno es usado a la vez, según el tipo de problema. Los sistemas operativos se construyen agregando capas de *software* (programas de sistema) a la máquina física del computador. APSE se construye, a su vez, agregando capas de software sobre el sistema operativo del computador anfitrión usado en Ada, como se ilustra en la figura 4. Usualmente se dibuja este tipo de diagramas en forma concéntrica, pero aquí se ha adoptado la forma indicada, para mayor claridad.

El nivel inferior es el del sistema operativo del computador anfitrión. El nivel siguiente es el del núcleo (*kernel*) del APSE, llamado KAPSE, cuyo objetivo es escudar al usuario de los detalles del sistema operativo y, por ende, del computador específico usado, lo que implica transformación lógica-física indispensable para asegurar portabilidad del programa, o independencia del APSE respecto a la máquina. Esto asegura también la reducción de gastos de entrenamiento de programadores (uno de los problemas actuales del DOD y de cualquier usuario de computadores). KAPSE no provee servicios directos al usuario y es relativamente transparente (oculto) para éste, pero proporciona ciertos soportes de biblioteca y de interfaz con periféricos y terminales.

FIG. 4. AMBITO DE SOPORTE DE PROGRAMACION ADA

El mínimo conjunto de herramientas necesarias para el desarrollo de *software* o programas es llamado MAPSE, y es representado como el nivel inmediatamente superior a KAPSE. MAPSE provee herramientas de *software* como editor de texto, editor de pantalla, formateador, compilador, enlazador, mantenedor de archivos, depurador, intérprete de comandos, y otros. Finalmente, el nivel superior es el APSE mismo, que proporciona herramientas más avanzadas que MAPSE, tales como aquellas para crear objetos de base de datos, modificaciones, transformaciones, despliegue, mantención, y otras.

El documento Stoneman deja bastante libertad en este nivel para futuros avances. En el APSE desarrollado por Rolm, con el nombre de ADE, antes mencionado, hay un conjunto de herramientas de *software* configuradas para soportar máquinas destinatarias específicas, el que incluye compiladores Ada, paquete de soporte de ejecución, ensamblador, enlazador,

importador y exportador. En particular, el importador es usado para traer al ADE (APSE) módulos producidos por otros compiladores (por ejemplo, Fortran 77) compatibles y llamables por el sistema operativo del anfitrión. El exportador permite formatear y transferir archivos de programas Ada del anfitrión al destinatario. La firma TeleSoft ha desarrollado un KAPSE con estricto cumplimiento a los estándares del lenguaje Ada, para varios tipos de computadores y sus sistemas operativos, y adaptable a futuros ámbitos Ada.

Comentarios finales

El lenguaje Ada ha sido desarrollado, vía un considerable esfuerzo internacional patrocinado y respaldado por el Departamento de Defensa de Estados Unidos, con las mejores técnicas modernas de la ingeniería de *software*, y muchas otras inéditas, que lo constituyen en el lenguaje computacional por excelencia en la década actual y otras venideras. Inicialmente ideado por el DOD para servir de lenguaje común en computadores incorporados en sistemas de defensa y armamentos, ha pasado a ser además, de gran importancia comercial y académica. Destinado originalmente a resolver el oneroso problema económico (de miles de millones de dólares) del DOD frente a la proliferación de lenguajes de programación (unos 450, incluidos los dialectos), ha servido de base para generar una importantísima industria de *software*, con cuantiosas sumas invertidas. De raíces asentadas en Pascal, está destinado a reemplazar, con ventajas notorias, a los conocidos lenguajes de alto nivel usados actualmente.

Dada la obligatoriedad de su uso en sistemas militares impuesta por el DOD, y la importancia económica de los contratos de ese Departamento, el lenguaje Ada se seguirá imponiendo en todos los ámbitos de la sociedad, trascendiendo su entorno original. Se dice en Estados Unidos que el programador que desee encontrar trabajo, ahora y a futuro, debe aprender Ada. Por el momento, como recién han aparecido compiladores y APSES validados, el lenguaje no se ha extendido mucho, pero lo empezará a hacer explosivamente desde el presente año (1985), por varias otras razones.

Ha habido antes otros experimentos para crear un lenguaje unificado o "esperantista" de programación, que —como en el caso de PL/I— no han sido exitosos. Algunos especialistas consideran que Ada debería tener exigencias más rigurosas aún, para convertirse en un lenguaje verdaderamente común. Temen otros que, por su complejidad, no resulte lo suficientemente confiable para uso en computadores que deben controlar sistemas muy importantes para la defensa y ofensa (misiles intercontinentales, submarinos nucleares, C3, bombas termonucleares, etc.) y que Ada pondrá más en peligro al mundo, por posibles errores. Pero, se contraarguye, los lenguajes y sistemas actuales de *software* en tales armas han demostrado fallas y errores, los que Ada permitirá evitar. El futuro de Ada es auspicioso y se espera que sea del agrado de la gran mayoría de los programadores actuales y venideros.

REFERENCIAS

1. U.S. DEPARTMENT OF DEFENSE-ADA JOINT PROGRAM OFFICE (DOD-AJPO): *Reference Manual for the Ada Programming Language*, DOD, Washington.
2. Computer (IEEE), June 1981. Edición dedicada al lenguaje Ada.
3. K.L. BOWLES: "Linked Ada Modules Shape Software Systems", *Electronic Design*, July 22, 1983.
4. I.C. PVLE: *The Ada Programming Language*, Prentice-Hall, 1981.

5. N. GEHANI: *Ada: An Advanced Introduction*, Prentice-Hall, 1983.
6. G. BOOCH; *Software Engineering with Ada*, Benjamin/Cummings, 1983
7. D.M. RUSSELL: "First Ada Compilers Show Diversity", *defense Electronics*, July 1983. pp. 35-36.
8. B. SHERMAN: ''Design of the First Ada KAPSE Interface", *defense Electronics*. April 1984. pp. 141-149.
9. D. KLEIN: A Buyer's a Guide to Ada Procurement", *Defense Electronics*, Junuary 1985, pp. 101-102.
10. JUAN L. HERNANDEZ S. "Empleo del lenguaje Ada en control de procesos industriales por publicar (1985).