Ada is Alive and Well in Air Traffic Management

Judith Klein

judith.klein@lmco.com

Outline

- Who we are, what systems we build
- Key Driving Requirements
- Key System Attributes
- Domain-independent infrastructure
- Evolving the infrastructure
- Domain-specific additions to infrastructure
- Ada - the preferred language
- Specific guidance for Ada usage
- Tools
- Some Ada lessons learned
- Transition to Ada95
- Conclusion

Lockheed Martin Air Traffic Management

- **Based in Rockville, Maryland, LMATM builds and integrates automation systems for use in air traffic management world wide**
  - En Route Host Computer System
  - Display Channel Complex Rehost
  - Host/Oceanic Computer System Replacement
  - *Republic of China Air Traffic Control Automation System (ATCAS)*
  - *En Route Display System Replacement*
  - *Minneapolis Community and Technical College (air traffic control training center)*
  - *United Kingdom New En Route Center*
  - Automated Radar Terminal System (and its derivatives) developed in Eagan, MN
  - SkyLine product featured in Area Control Center for Korea (Inchon Airport) under development

developed primarily in Ada

Key Driving Requirements

- **Fault tolerance/Reliability**
  - 10 seconds to recover from some interruptions
  - Error detection, containment, localization
  - Error recovery
- **Continuous operations (24x7)**
  - Memory management
    - Memory leaks cannot be tolerated in 24x7 operations
  - Distribution of new releases during systems operation
    - Cannot completely shut down ATC operations during system software upgrades
    - Support separate testing and training subsystem for new releases
    - Facilitate speedy cutover to the newly distributed/tested release
  - On-line hardware certification, monitoring (facilitate maintenance)
- **Performance monitoring**
- **System recording**
  - Off-line problem determination
  - Incident analysis
  - System health verification
  - Workload analysis

Key System Attributes to Address Driving Requirements

- **Theme: redundancy, independence of failures, automated recovery**
- **Independent subsystems**
  - For use during planned system outage for maintenance
  - For use as backup either for one controller position or for an entire subsystem (in case of catastrophic failure)
- **Distributed System Architecture**
- **Within a subsystem, primary and backup servers**
  - Use of redundancy to achieve recovery in 10 seconds
- **Within a node, individually startable/stoppable programs**
  - Fail independently of each other
  - Limit scope of lost function
- **Availability Management Hierarchy**
  - Detect errors at the lowest level possible
  - Isolate and recover automatically whenever it is safe to do so
- **Problem Determination Architecture**
  - Gather on-line, analyze off-line
  - Capture information when problem occurs

Domain-independent Infrastructure

- **A rich set of services provided**
  - Communication with other applications (location, and primary vs. standby status, are transparent);
  - Recording errors, debug data, state data
    - Recording architecture supports continuous and event driven recording
  - Capturing elements of end-to-end system threads for reporting of response times
- **Common, reusable application framework defined**
- **Many "templates" of correct behavior developed for use by application developers**
  - What data to checkpoint, how to handle bad checkpoint files
  - What data to keep "hot", when to copy it to the standby
- **Some "templates" are tailorable, some are Ada generics, others are adapted**
- **Types Dictionary Service**
  - Describes data layout
  - De-couples on-line and off-line (e.g., adaptation generation, analysis) software

Infrastructure Reused

- **First operational use was for Republic of China**
  - A single En Route center supporting six controller workstations and three approach centers with six controller workstations each
- **"Display System Replacement" is the largest scale user of the infrastructure**
  - The twenty locations vary in size, with the largest center supporting more than 130 controllers with over 200 processors. Replaces display channels, plan view displays, etc.
- **New En Route Center infrastructure is a close derivative of the US one**
  - The two systems are benefiting from each other's enhancements, fixes

- **Infrastructure is about 250,000 SLOC**
- **Display System Replacement for US En Route centers is about 500,000 SLOC**
- **New En Route Center for United Kingdom is about 1,000,000 SLOC**

Evolving the Infrastructure

- **Software for next generation programs is in various stages of prototyping, porting**
- **Extending services**
  - Additions to the domain-independent infrastructure
  - Initial set of air traffic management services (i.e., domain-specific) published
- **Infrastructure Application Programming Interface further abstracted and simplified**
  - Consolidated services
  - Facilitated platform independence for applications
  - Utilized industry standard interfaces (as available) to insure ease of portability for infrastructure itself
- **Porting the infrastructure to Ada95 for use in future programs**
  - Trade study conducted to select an Ada compiler for Sun/Solaris platform; GNAT chosen

Technical Requirements for Ada95 Compiler

- **Current needs for Ada95 compiler dictate the following**
  - Product must execute on and produce output executable on a Sun SPARC processor running Solaris 2.6
  - Product must be a validated Ada95 compiler at the time of evaluation
  - Product must support interfacing to C, C++ and assembly language
  - Product must support implementing Ada tasks as Operating System dispatchable entities (threads or processes)
  - Provide a runtime locking mechanism that will prevent priority inversion
  - Product must not impose runtime license requirements on generated executables which would involve license servers or other overhead to verify licensing restrictions
  - Product must support or plan to support ASIS (Ada Semantic Interface Specification) 2.0 or equivalent functionality. ASIS 2.0 is strongly preferred to any equivalent functionality. ASIS supports generation of type descriptor dictionaries, which are used in the generation of adaptation and preset data. ASIS is also used in the Ada code standards checker.
  - Provide the ability to trace back through a chain of procedure or function calls and provide a callable application interface (not a debugger) that returns the exception traceback data, formatted in ASCII and indicating the unit and line number where the call occurred
  - Support an enhanced assembler listing to show the record type layout information
  - Allow presence of floating point components in a record without imposing limitations on the size of the record nor on the alignment of those components within the record

Reference

Domain-specific Additions to Infrastructure

- **Flight Data Application Programming Interface**

- **Facilitates commonality between implementations of strategic Air Traffic Control applications; some common objects of interest are:**
  - Flight plans, Converted routes, Trajectories, Conflicts, Airspace definition, Routes, Gridded winds, pressure and temperature, etc.
- **Facilitates a common implementation of data and services for multiple flight data applications**
- **Hides the implementation details of data**
  - Minimizes application impacts when data structures are modified
- **Hides distribution details of data**
  - Applications are only concerned with what data they own, manage, or subscribe to
  - Applications are not concerned with the origin and ownership of data they subscribe to, nor with the consumers of data they publish

Ada - the Preferred Language

- **International Safety Standard IEC-1508**
  - Recommends languages for systems on different “safety integrity levels”
  - Typically, Air Traffic Control systems are Safety Integrity Level 3 (probability of failure between 10^{-3} and 10^{-4}) or Safety Integrity Level 4 (between 10^{-4} and 10^{-5})
  - For these Safety Integrity Levels
    - Ada is “Highly Recommended”
    - C is specifically “Not Recommended”
    - C with a subset defined and a coding standard in place is neither “recommended” nor “not recommended”
- **Development environment, expertise in place**
  - Design tool, customized notation convention
  - Style Supplement (guidance for use of language features)
  - Tools (e.g., standards checker)
  - Team core has Ada development and debugging expertise

Ada95 Preferred for New Development

- **Does not force design methodology (Object Oriented or Function Oriented), yet supports both**
- **The first internationally-standardized, fully object-oriented programming language**
- **Meets sophisticated and complex requirements for modern software engineering methodology**
  - Modularization
  - Information hiding
  - Inheritance
  - Reuse through separate packages and generics
- **Supports fundamental safety features, such as**
  - Run-time array bound checking
  - Run-time parameter checking
  - Exception handling
  - Information hiding
  - Variable sub-range definition

Ada83 - the Legacy Language of Choice

- **Ada used wherever possible in the operational system**
- **Some C, scripts, shell was used where Ada not appropriate**
- **OC Systems Ada83 (Legacy Ada) compiler**
  - Custom heap manager (including “tracker”) implemented through the use of commercial API (performance and reliability considerations)
  - Custom lock package provides protection from priority inversion (priority inheritance protocol)
  - Dynamic call chain traceback as well as Exception traceback information provided by OC Systems “Legacy Ada” compiler through a custom interface
    - We found this very useful for recording the dynamic location of detected error conditions to help with debugging
- **Ada task mapped to an AIX process**
  - Each process can have a different priority

Ada Features Used/Prohibited

- **Extensive guidance on exception handling; examples:**
  - Recording all propagated exceptions in package specification prolog
  - Use exceptions for exceptional conditions (rather than 'goto')
  - Must have a "when others" handler in the Main module
  - Beware of code executed in exception handlers to insure that the most severe exception is the one ultimately propagated
  - User-defined exceptions should be handled explicitly by name
  - Don't raise Ada predefined exceptions explicitly
  - Locate handling of Ada predefined exception as closely as possible to point of raise
  - Package body should not propagate locally-defined exceptions (anonymous exception would be propagated)
- **Use of some "pragma"s prohibited**
  - "Controlled", "Memory_Size", "Optimize", "Shared", "Storage_Unit", "System_Name" - all prohibited; compiler used does not support some of them, for some similar capability provided in an alternate way
- **Data validation techniques recommended**
  - Ada95 will help here; in Ada83, following use of Unchecked_Conversion, validation code was non-trivial because simple checks are optimized out by the compiler

Software Productivity Consortium (SPC)
Ada Style Guide Customized

- **Naming Conventions**
  - "Do not use identifiers constructions (e.g., suffixes) that are unique to (sub)type identifiers" - we suffix types with _T
- **Tasks**
  - Usage restricted for 'delay', 'rendezvous' (performance concern)
- **Anonymous Types, Anonymous Task Types**
  - "Use anonymous types for array variables when no suitable type exists and the array will not be referenced as a whole" - we avoid anonymous types in all cases
  - "Use anonymous task type for single instances" - because the use of anonymous task type precludes the specification of a Storage_Size length clause we require specification of task types
- **Shared Variables and Pragma Shared**
  - "Do not share variables", and "Do not use shared variables as a task synchronization device" - we allow sharing when accompanied by appropriate access synchronization
  - "Have tasks communicate through the rendezvous mechanism" - we provide a service for queuing between tasks and disallow rendezvous in applications so we can monitor tasks for health
- **Coupling Due to Pragmas**
  - "For non-generics named in a context clause, avoid pragma Elaborate" - we use it extensively to avoid 'Program_Error' and for performance gain (eliminates costly elaboration checks) and reduced variability in program behavior over time

Tools

- **TeamworkAda for static (architecture) diagrams, threads of execution (system-wide and within an application)**
- **Standards checker, code formatter**
- **McCabe complexity analyzer**
- **OC Systems Aprobe for real-time debugging in a "non-interfering" manner**
  - Useful for unit testing as well (e.g., raise specific exceptions, force error paths, verify coverage)
- **AIX trace for debugging O/S process scheduling problems**
- **AIX trace also used for measuring path lengths, which are then used in system modeling for adjusting performance allocations**
- **vmstat - monitor CPU utilization and memory**

More Tools

- **Static Analysis Tools to assist in memory analysis**
  - ***asizes*** - runs against an executable file, gives size of CSECTs and static data areas
    - works by simply dumping symbol table and calculating difference between symbol offsets
  - ***aframes*** - runs against an executable file, provides stack frame sizes, location of dynamic stack extensions, location/size (when static) of heap allocations
  - ***assembly listings*** - provide detailed stack frame or static data area layout by variable
- **Moving to a new environment and leaving behind the known tools**
  - Ada95 compiler differences
  - Standards checker will have to be updated for Ada95
  - Exception traceback - changing to use the Ada.Exceptions standard interface, but
    - the data contents is not defined; some vendors don't implement, or return HEX Program Counter values vs. unit, line number information

Some Ada Lessons Learned

- **Examples based on our experience (compiler, HW, system load combination)**
  - For a more complete discussion on the subject see Chapter 10 in SPC's *Ada 95 Quality and Style Guide: Guidelines for Professional Programmers*
- **Excessive data movement**
  - Functions returning large composites
    - Use procedure with 'out' parameter to pass by reference instead of copied onto the stack - compiler implementation dependent!
  - Aggregate assignments
  - Simple assignment of large structures
  - Unnecessary temporary variables
  - Initialization of data structures
  - Default on types and on procedure parameters
- **Excessive subprogram calling overhead (too many subprograms without good rationale)**
  - Could 'inline', could remove defaults, could redesign/recode
  - Avoid long chains of inheritance/object extension in Ada95
- **Using exceptions in non-exceptional situations**
  - If not really exceptional, use a boolean or enumeration return code

Ada95 Features we Missed in Ada83

- **Obtaining a pointer to a statically allocated variable**
  - we used *'Address* and *Unchecked_Conversion* between *System.Address* and pointer type
- **Task priorities**
  - we mapped an Ada task to an OS process, then infrastructure assigned process priorities during initialization
- **Data validation techniques, X'Valid**
- **Shared Variable Control**
- **Child packages**
- **Access procedure, P'Access**

Conclusion

- **Ada83 used on several large, distributed, real-time, highly available systems for use in air traffic management**
- **Ada95 is being used in the evolution of these systems**
- **All the original reasons for selecting Ada remain valid: Ada is the best language in real-time, safety-critical domains**