Ada's approach to Software Vulnerabilities

Stephen Michell
Maurya Software
Ottawa Canada

25/06/06

Outline

- Introduction, History
- Rationale
- Approach
- Language Analysis
- Assessment of Approach

Introduction

- Developed by High Integrity Rapporteur Group of SC22/WG9 (Ada) to address specific needs of Safety and Security communities
- Coined the term High Integrity to represent them

Introduction (cont)

- Based on **Requirements** for software development and verification of various standards
  - Airborne Civil Aeronautics(DO-178B)
  - Nuclear Power (IEC880) (NRC)
  - Medical Systems (IEC 601-4) (FDA)
  - UK Defence (DS 00-55)
  - European Security (ITSEC)
  - European Rail (EN50128)
  - UK Automotive (MISRA)
  - Space (NASA)

Introduction (cont)

- Developed a framework for analysis of software
  - Started with examination of current common verification techniques
- Developed analysis approach to analyse the appropriateness of language features

Introduction (cont)

- Always come back to requirements of community that needs this
- Why?
  - If they can't get the information, they will use something else
  - Stops language-war or methodology-war arguments
    - You need to show how XXX satisfies requirement YYY

Introduction (cont)

- Observation about term “vulnerabilities”
  - Really a negative term
  - All software is vulnerable
    - Assembler the most
  - Modern languages already impose restrictions on the way that we can express certain concepts and check that the usage was proper
  - Really guidance on the use of language features to enhance verifiability
  - Need a positive spin on what we produce

History of Guidance on use of Ada in HI Systems

- Began as Ada9X project was publishing Ada95
- Number of Software-related safety documents being developed/published
  - UK DIS 0055/56
  - MISRA
  - CAC/SEC
- Ada 83 (subsets) being used in HI systems,
- Ada95 added many newcapabilities

History of Guidance on use of Ada in HI Systems

- WG9 formed HRG to address needs of Safety and Security communities
- Canadian study (with HRG input) developed Framework for Analysis, feature x feature analysis, initial ratings
- HRG took work
  - Extended (tasking, exceptions, generics)
  - Condensed (tabular form)
  - Published as TR15942 Guidance on the use of the Ada Programming Language for High Integrity Systems

History of Guidance on use of Ada in HI Systems

- In use by many organizations to support their HI development
  - Aviation
  - Rail
  - Space
  - Nuclear

Rationale

- Language features add capability
  - Expressability
  - Better conceptualization
  - Better human communication
- Features may have high implementation, usage or verification costs.

Rationale (cont)

- Why not just use tools?
  - Ada syntax straightforward
  - Ada semantics (overloading, overriding, name resolution) beyond most simple tools
  - Tools often misused
    - Ignored when most needed (eg systems integration)
    - Tool misinterpretation or extra requirements

Verification Techniques

- TR defined approaches required by HI standards
  - Traceability
  - Reviews
  - Analysis
  - Testing

Verification Techniques (cont)

- Traceability
  - Requirements<->requirements
  - Requirements -> design, code, test
  - Object code -> source code
- Reviews
  - Human based
  - Formal or informal
  - **Independent**

Verification Techniques (cont)

- Traceability and Reviews human activities
  - Ones that include language-specific or tools (for repeatability) included in “analysis”
    - Rest left out of scope

Verification Techniques (cont)

- Analysis
  - Static
    - Control Flow / Data Flow
    - Information Flow / Symbolic Execution
    - Formal Code Verif / Range Checking
    - Stack Usage / Timing Analysis
    - Other Mem Usage / Object Code Analysis

Verification Techniques (cont)

- Dynamic (Testing)
  - Levels
    - Unit
    - Integration
    - Hw/Sw integration
    - System
  - Types
    - Requirements-Based Testing
    - Structure-Based testing

Language Analysis(cont)

- Nine categories captured
  - Flow Analysis(FA)
  - Symbolic Analysis(SA)
  - Range Checking(RC)
  - Stack Usage(SU)
  - Timing Analysis(TA)
  - Other Memory Usage(OMU)
  - Object Code Analysis(OMA)
  - Requirements-based Testing(RT)
  - Structure-based Testing(ST)

Language Analysis(cont)

- Provides a 3-way categorization to capture the applicability of language features viz-a-viz the analysis categories

Language Analysis(cont)

- Three guidance categories
  - Inc – Included -
    - Directly amenable to analysis technique
  - Alld – Allowed
    - Technique not straightforward but achievable OR
    - Use of feature needed and problems in verification technique can be circumvented
  - Exc – Excluded
    - No current cost effective analysis technique
    - Projects should have ways to ensure capability is excluded

Language Analysis(cont)

- Discussion of a number of issues in writing verifiable programs
  - How language rules
    - Affect Predicatbility
    - Support Modelling
    - Facilitate Testing

Language Analysis(cont)

- Language devided into 14 sets of features
  - Types with static attributes
  - Declarations
  - Names
  - Types with Dynamic att's
  - Statements
  - Subprograms
  - Packages
  - Arithmetic Types
  - Low level/Interfacing
  - Generics
  - Expressions
  - Exceptions
  - Tasking
  - Distribution

Language Analysis(cont)

- Example – Types with Dynamic Attributes
  - Introduction
    - Most unconstrained types have constrained objects
    - Unconstrained parameters have constrained actuals
    - Access types more secure than many languages, but must avoid heap and watch aliasing
    - Storage pool preferrable to Heap
    - Run-time sizing of objects makes bounding storage use difficult
    - No variant records

Language Analysis(cont)

Types with Dynamic Attributes

Feature	FA	SA	RC	SU	TA	OMU	OCA	RT	ST
Unconstrained array types - including strings[1]	Inc	Inc	Inc	Inc	Inc	Inc	Inc	Inc	Inc
Full access types	**Exc**[2]	**Exc**[2]	Inc	**Exc**[2]	**Exc**[2]	**Exc**[2]	Inc	Inc	Inc
Restricted storage pools[3]	**Alld**[4]	**Alld**[4]	Inc	Inc	Inc	Inc	Inc	Inc	Inc
General access types	**Exc**[4]	**Exc**[4]	Inc	Inc	Inc	Inc	Inc	Inc	Inc
Access to subprogram	**Exc**[5]	**Exc**[5]	Inc	Inc	**Alld**[5]	Inc	**Alld**[5]	Inc	Inc
Controlled types including unrestricted storage pools	**Exc**[6]	**Exc**[6]	Inc	Inc	Inc	Inc	**Alld**[6]	Inc	**Alld**[6]
Indefinite objects[7]	**Alld**	**Alld**	**Alld**		**Exc**	**Exc**	**Exc**	Inc	**Exc**
Non-static array objects[8]	Inc	**Alld**	**Alld**	**Alld**	**Alld**	**Alld**	Inc	**Alld**	Inc

Types with Dynamic Attributes

- Notes
  - 1. reference sect 5.6,Concatenate fn returns UT
  - 2. Full access types use runtime heap, Mem use, TA problematic, fragmentation. Risk of unbounded aliasing
  - 3. Pool-specific access types similar to stack-based types, watch implementation
  - 4. Aliasing problem
  - 5. CFA, TA intractible
  - 6. Hidden control flows in controlled ttypes
  - 7. TA, SA of indefinite objects unpredictable
  - 8. TA, SA of run-time dynamically bound types

Types with Dynamic Attributes

- Guidance
  - Language-provided restrictions
    - No_Implicit_Heap_Allocation
    - No_Allocators
    - No_Access_Subprograms
  - Caution on “Inc” features – risk of aliasing, difficulties of review

Assessment of Approach

- Imtimately tied with technology at time of writing
  - Verification approaches mature with time.
- Considered but doesn't explicitly express some of the concerns being considered here, such as attack modes

WG9's next steps

- Current guidelines published and in significant use for 6 years
- SC22 documents likely to be many years from publication, and quality unknown
- Eager to participate but don't want to delay own work
- Hope new insights and approaches may come from analysis in larger arena

WG9's next steps (cont)

- Currently WG9/HRG looking to revise existing document
  - HI Standards have changed
  - New analysis techniques
  - New threats
  - New analysis techniques for some language features may change approach (concurrency, OO)
  - New language features (Interfaces)