Before that...

- Last week I was at a workshop on highly dependable computing systems
  - At NASA Ames Research Center
- Academia, government, industry
  - IBM, Sun, Oracle, Sybase, Microsoft, Boeing, Honeywell, ...
- Keynotes, case studies, breakout sessions, etc.
- Dependability is different things to different people
- Over all, I think that there were two camps
  - Use technology to improve dependability
  - Build a “culture of dependability”

Two NASA failures: each over $100 million
http://www.nasa.gov/newsinfo/marsreports.html

- Mars Climate Orbiter
  - A confusion in metric/English units caused an engine to fire too strongly, bringing the spacecraft too close to Mars and causing it to crash instead of orbiting
- Polar lander (very probable cause)
  - At 40m above Mars, a parachute and spring-loaded legs were deployed and a new control regime was used as planned
  - The spring-loaded legs bounced, causing the regime to think that the pads had hit the surface
  - The engine was turned off and the spacecraft crashed

Specification errors?

- Not the units
  - The specification was completely clear about this
  - A new programmer didn’t know or check, and used the wrong units
  - Not caught by testing, inspections, etc.
    - Tricky to catch by testing, since it was a second order effect
  - What can be done about errors like these?
- Polar lander? Unclear
  - Each module (regime) worked as specified
  - The <40m module assumed that a variable would be in a particular state upon entry, but it wasn’t due to the leg bounce
  - What this a problem in the inter-module specification? In the implementation of the <40m module? Testing?

Specifications thread

- I found it interesting to come back from this workshop and see the thread on the mailing list about “spec avoidance”
- Specs would surely help solve some — perhaps many — of your problems
  - But not all
  - And the cost is not clear
  - I’ll note that most of you wanted specs, but didn’t necessarily want the responsibility of writing them
- See the paper that Cordell Green mentioned, which I posted on the mailing list
State machines

- Good for specifying reactive systems, protocols, etc.
- Event-driven
  - External events (actions in the external environment, such as “button pushed”, “door opened”, “nuclear core above safe temperature”, etc.)
  - Internal events (actions defined in the internal system to cause needed actions)
  - Can generate external events that may drive actuators in the environment (valves may be opened, alarms may be rung, etc.)
  - Transitions can have guards and conditions that control whether or not they are taken
- “Flat” (non-hierarchical) state machines tend to explode in size relatively quickly

Classic examples

- Specifying a cruise control
- Specifying the traffic lights at an intersection
- Specifying trains on shared tracks
  - Could be managing the bus tunnel in Seattle
- Etc.

A snippet of cruise control

- OnButtonPushed
- Cruise
- Pause
- OffButtonPushed
- Exceed25MPH
-…
- LockButtonPushed

More cruise control

- What if your state machine also tracked speed?
  - Maybe the cruise control doesn’t work at low speeds
  - Anyway, it needs to remember a speed so it can resume properly
- What if it also interacted with the door locking system?
  - You might have to modify almost every state to track not only the state on the previous slide, but the speed, too
  - Essentially, you need to build a cross product of all combinations of states
- This is the kind of issue that can cause the machine to blowup in size
  - It’s not the best example, but it’s adequate

Statecharts: “review”

- The idea of statecharts [Harel] is to provide a rich, visual representation for defining finite state machines that capture the essence of complex, reactive systems
- Specifically addresses description explosion problem
- Sorry, there isn’t a simple, easy-to-get reference
- The i-Logix web site (http://www.ilogix.com/fs_papers.htm) has a set of papers, (you have to register your name, but it looks like it doesn’t check anything)

Key idea: hierarchy

Exceed25MPH
-…
LockButtonPushed
-25MPH

OnButtonPushed
Cruise
Pause
Exceed25MPH
-…
OffButtonPushed

Parallel AND-machines

- The state of the overall machine is represented by one state from each of the parallel AND machines
  - In a cruise control state AND in a speed state AND in a door lock state
- Transitions can take place in all substates in parallel
  - Events in one substate can cause transitions in another substate

A few statechart features

- Default entry states for each substate
  - Indicated by an arrow with no initial state
- When any of the parallel machines is exited, the entire machine is exited
- You can have “history” states, which remember where you were the last time you were in a machine
  - The “STATEMATE semantics” are the standard semantics
  - This is largely a question of which enabled transitions are taken, and when
  - At this level, you surely don’t care

Variants on statecharts

- There are many variants on statecharts
- One is RSML (Leveson et al.), which allows states to be connected through a bus as well as pairwise
- RSML also represents transitions differently, through explicit AND-OR tables instead of through guards and conditions on transitions

Sample transitions

- On
  - Trigger_Event: Temp_Update
  - Condition: Temperature in Too Hot
  - Output Action: Panic_Event

  - Ready
  - Move In
    - Trigger_Event: Temp_Update
    - Condition: Rod Movement in Ready and Temperature in Hot
    - Output Action: Initiate_Move

  - Just Moved
    - Trigger_Event: Clock_Event
    - Condition: Rod Moved in Just Moved and t > entered(Just_Moved) + Move_Delay

Leap of faith

- Statecharts (and variants) can be used to specify important, complex systems
Question

- So we have a big statecharts-like specification
- How do we know it has properties we want it to have?
  - Ex: Is it deterministic?
  - Ex: can you ever have the doors unlock by themselves while the car is moving?
  - Ex: can you ever cause an emergency descent when you are under 500 feet above ground level?

Standard answers include

- Human inspection
- Simulation
- Analysis

An alternative: model checking

- Evaluate temporal properties of finite state systems
  - Guarantee a property is true or return a counterexample
  - Ex: Is it true that we can never enter an error state?
  - Ex: Are we able to handle a reset from any state?
- Extremely successfully for hardware verification
  - Intel got into the game after the FDIV error
- Open question: applicable to software specifications?

State Transition Graph

- One way to represent a finite state machine is as a state transition graph
  - S is a finite set of states
  - R is a binary relation that defines the possible transitions between states in S
  - P is a function that assigns atomic propositions to each state in S
    - e.g., that a specific process holds a lock
- Other representations include regular expressions, etc.

Example

- Three states
- Transitions as shown
- Atomic properties a, b and c
- Given a start state, you can consider legal paths through the state machine

A computation tree

- From a given start state, you can represent all possible paths with an infinite computation tree
- Model checking allows us to answer questions about this tree structure
Temporal formulae

- Temporal logics allow us to say things like:
  - Does some property hold true globally?
  - Does some property inevitably hold true?
  - Does some property potentially hold true?

Mutual exclusion example

- N1 & N2, non-critical regions of Process 1 and 2
- T1 & T2, trying regions
- C1 and C2, critical regions
- AF(C1) in lightly shaded state?
  - C1 always inevitably true?
- EF(C1 ∧ C2) in dark shaded state?
  - C1 and C2 eventually true?

How does model checking work? (in brief)

- An iterative algorithm that labels states in the transition graph with formulae known to be true
- For a query Q
  - the first iteration marks all subformulae of Q of length 1
  - the second iteration marks them of length 2
  - this terminates since the formula is finite
- The details of the logic indeed matter
  - But not at this level of description

Example

- Q = T1 ⇒ AF C1
  - If Process 1 is trying to acquire the mutex, then it is inevitably true it will get it sometime
- Q = ¬T1 ∨ AF C1
  - Rewriting with DeMorgan’s Laws
- First, label all the states where T1, ¬T1, and C1 are true
  - These are atomic properties

Example

- Next mark all the states in which AF C1 is true, etc.
  - The algorithm tracks states visited using depth-first search
  - Slight variations for AF, AG, EF, EG, etc.
- At termination, ¬T1 ∨ AF C1 is true everywhere
  - Hence the temporal property is true for the state machine
Binary decision diagrams (BDDs)

- "Folded decision tree"
- Fixed variable order
- Many functions have small BDDs
  - Multiplication is a notable exception
- Can represent
  - State machines (transition functions) and
  - Temporal queries

Due to Randy Bryant

BDD-based model checking

- Iterative, fixed-point algorithms that are quite similar to those in explicit model checking
- Applying boolean functions to BDDs is efficient, which makes the underlying algorithms efficient
  - \( \land \) becomes set intersection, \( \lor \) becomes set union, etc.
- When the BDDs remain small, that is
  - Variable ordering is a key issue

BDD-based successes in HW

- IEEE Futurebus+ cache coherence protocol
- Control protocol for Philips stereo components
- ISDN User Part Protocol
- ...

Software model checking

- Finite state software specifications
  - Reactive systems (avionics, automotive, etc.)
  - Hierarchical state machine specifications
- Not intended to help with proving consistency of specification and implementation
  - Rather, checking properties of the specification itself

Why might it fail?

- Software is often specified with infinite state descriptions
- Software specifications may be structured differently from hardware specifications
  - Hierarchy
  - Representations and algorithms for model checking may not scale

Our approach at UW—try it!

- Applied model checking to the specification of TCAS II
  - Traffic Alert and Collision Avoidance System
  - In use on U.S. commercial aircraft
  - http://www.faa.gov/and/and600/and820/newtcas.htm
  - FAA adopted specification
  - Initial design and development by Leveson et al.
- Later applied it to a statecharts description of an electrical power distribution system model of the B777
  - I can provide examples and papers
TCAS

- Warn pilots of traffic
  - Plane to plane, not through ground controller
  - On essentially all commercial aircraft
- Issue resolution advisories only
  - Vertical resolution only
  - Relies on transponder data

TCAS specification

- Irvine Safety Group (Leveson et al.)
  - Specified in RSML as a research project
    - RSML is in the Statecharts family of hierarchical state machine description languages
    - FAA adopted RSML version as official
- Specification is about 400 pages long
  - This study uses: Version 6.00, March 1993
    - Not the current FAA version

TCAS—high-level structure

- Own_Aircraft
  - Sensitivity levels, Alt_Layer, Advisory_Status
- Other_Aircraft
  - Tracked, Intruder_State, Range_Test, Crossing, Sense Descend/Climb

Using SMV

- SMV is a BDD-based model checker
- It checks CTL formulas
  - A specific temporal logic

Iterative process

- Iterate SMV version of specification
- Clarify and refine temporal formula
- Model environment more precisely
- Refine specification

Use of non-determinism

- Inputs from environment
  - Altitude := {1000..8000}
- Simplification of functions
  - Alt_Rate := 0.25*(Alt_Baro-ZP)/Delta_t
  - Alt_Rate := {-2000..2000}
- Unmodelled parts of specification
  - States of Other_Aircraft treated as non-deterministic input variables
Translating RSML to SMV

MODULE main
VAR
  state:{ON, OFF};
  on_event: boolean;
  off_event: boolean;
ASSIGN
  init(state) := OFF;
  next(state) := case
    state = ON &
    off_event: OFF;
    state = OFF &
    on_event: ON;
  1 : state;
esec;

State encoding

• Flatten nested AND
  and nested OR states
• One variable for each
  OR state
  – An enumerated type
    of the alternatives

VAR
  S: {A, B, C};
  T: {D, E};
  U: {F, G};

Events

• External—interactions with
  environment
• Internal—micro steps
• Synchrony hypothesis
  – External event arrives
  – Triggers cascade of internal events (micro
    steps)
  – Stability reached before next external
    event
• Technical issues with micro steps

Non-deterministic transitions

• A machine is deterministic if at
  most one of $T_{A \rightarrow B}$, $T_{A \rightarrow C}$, etc. can
  be true
  – $T_{A \rightarrow B}$ represents the conditions
    under which a transition is taken
    from state $A$ to state $B$
  – Else non-deterministic

Checking properties

• Initial attempts to check any property
  generated BDDs of over 200MB
• First successful check took 13 hours
  – Was reduced to a few minutes
• Partitioned BDDs
• Reordered variables
• Implemented better search for
  counterexamples

Property checking

• Domain independent properties
  – Deterministic state transitions
  – Function consistency
• Domain dependent
  – Output agreement
  – Safety properties
• We used SMV to investigate some of
  these properties on TCAS’
  Own_Aircraft module
Disclaimer

The intent of this work was to evaluate symbolic model checking of state-based specifications, not to evaluate the TCAS II specification. Our study used a preliminary version of the specification, version 6.00, dated March, 1993. We did not have access to later versions, so we do not know if the issues identified here are present in later versions.

Deterministic transitions

- Do the same conditions allow for non-deterministic transitions?
- Inconsistencies were found earlier by other methods [Heimdahl and Leveson]
  - Identical conditions allowed transitions from Sensitivity Level 4 to SL 2 or to SL 5
- Our formulae checked for all possible non-determinism; we found this case, too

Note: Earlier version of TCAS spec

Function consistency

- Many functions are defined in terms of cases
- A function is inconsistent if two different conditions $C_i$ and $C_j$ and be true simultaneously

$$F = \begin{cases}
  V_1 & \text{if } C_1 \\
  V_2 & \text{if } C_2 \\
  V_3 & \text{if } C_3
\end{cases}$$

$$\mathcal{AG} \neg ((C_i \land C_j))$$

Display Model Goal

- Tells pilot desired rate of altitude change
- Checking for consistency gave a counterexample
  - Other Aircraft reverse from an Increase-Climb to an Increase-Descend advisory
  - After study, this is only permitted in our non-deterministic modeling of Other Aircraft
  - Modeling a piece of Other Aircraft's logic precludes this counterexample
Output agreement

- Related outputs should be consistent
  - Resolution advisory
    - Increase-Climb, Climb, Descend, Increase-Descend
  - Display_Model_Goal
    - Desired rate of altitude change
    - Between -3000 ft/min and 3000 ft/min
  - Presumably, on a climb advisory, Display_Model_Goal should be positive

Output agreement check

- AG (RA = Climb -> DMG > 0)
  - If Resolution Advisory is Climb, then Display_Model_Goal is positive
- Counterexample was found
  - t_1: RA = Descend, DMG = -1500
  - t_2: RA = Increase-Descend, DMG = -2500
  - t_3: RA = Climb, DMG = -1500

Limitations

- Can’t model all of TCAS
  - Pushing limits of SMV (more than 200 bit variables is problematic)
  - Need some non-linear arithmetic to model parts of Other_Aircraft
    - New result that represents constraints as BDD variables and uses a constraint solver
  - How to pick appropriate formulae to check?

Whence formulae?

- Jaffe, Leveson et al. developed criteria that specifications of embedded real-time systems should satisfy, including:
  - All information from sensors should be used
  - Behavior before startup, after shutdown and during off-line processing should be specified
  - Every state must have a transition defined for every possible input (including timeouts)
    - Predicates on the transitions must yield deterministic behavior
  - Essentially a check-list, but a very useful one

What about infinite state?

- Model checking does not apply to infinite state specifications
  - The iterative algorithm will not reach a fixpoint
- Theorem proving applies well to infinite state specifications, but has generally proved to be unsatisfactory in practice
  - One approach is to abstract infinite state specifications into finite state ones
    - Doing this while preserving properties is hard
- D. Jackson et al.’s Nitpick approach
  - Find counterexamples (errors), but don’t “prove” anything
Model checking wrap up

- The goal of model checking is to allow finite state descriptions to be analyzed and shown to have particular desirable properties
- Won’t help when you don’t want or need finite state descriptions
- Definitely added value when you do, but it’s not turnkey yet
- There’s still a real art in managing model checking
- Definitely feasible on modest sized systems

I know this was quick

- My goal isn’t to make you into model checking experts
  - But it might titillate one or two of you to learn more
- But rather to understand the sketches of what model checking is and why it is so promising for checking some classes of specifications

It’s show time!

- Michael Jackson’s keynote address at the 17th International Conference on Software Engineering (ICSE 17)
  - 1000 researchers, educators, and practitioners
  - Other keynoters: Fred Brooks, Michael Cusamano
- Discussion on the mailing list...

Ways of Looking at Software

- “Programming should be literate”
- “… they regarded my program as logical poems …”
- “The goal of any system is organizational change”
- “Software development is engineering”
- Because we make machines to serve useful purposes in the world
  - The problem is in the World
  - The Machine is the solution

WHAT and HOW

- WHAT does an automobile do?
  - It carries people and their luggage, travelling over roads where its driver directs it to go
- WHAT is in the world, HOW is in the machine
The Machine, the Model, and the World

- Formal Methods concern the left arrow
- We have no theory for the right arrow
Brian Cantwell Smith, The Limits of Correctness

Talking about the World and the Machine

- To develop software we must talk both about the World and about the Machine
- But it's hard to maintain the right balance between these two outlooks of discourse
- The relationship between them is varied and often subtle
- Often we have personal preferences to exploit or resist

Three Topics and a Button

- 4 Facets of the Relationship
- 4 Kinds of Denial of the World
- 4 Principles for Accepting the World
- a Button

4 Facets of the Relationship

- Modelling: the Machine as a model of the World
- Interface: what the Machine shares with the World
- Engineering: how the Machine changes the World
- Problem: the structure the Machine must have to fit the problem in the World

Modelling a Reality

- "An SADT system definition is called a "model" ..."
- R L Ackoff (Scientific Method, 1962):
  - Iconic models — pictures, 3-D representations, eg a child's model farm
  - Analytic models — manipulable formal descriptions, eg differential equations forming an economic model
  - Analogic models — an analogous reality, eg an electrical network modelling the flow of water in pipes
  - Software models are analogic, eg, a database, an assemblage of objects, a process network

The Machine As a Model of the World

- Authors, Books, PublishedBy
- A Books, N-Records, PublishTo
Modelling and (Ω)
- A data model fragment:
  - Descriptions of the World
  - Descriptions of the Machine
- Three sets of descriptions:
  - Name
  - Birth
  - Author
  - Publisher
  - Date

Non-Modelling and (Ω)
- Both the World and the Machine have properties that are private and not shared
  - Record Deletion
  - Normalisation
  - Record Sequencing
  - Null Field Values
  - Multiple Authors
  - Anonymous Works
  - Multiple Phasers
  - Linked Novels

The Machine – World Interface
- Shared phenomena: events, other shared individuals, faces visible in both domains
- No communication without sharing:
  - Communication
  - In 'really'
  - Shared event
  - Shared event 'not shared'

Shared Phenomena
- Shared phenomena:
  - Issues
  - High events
  - HighOff events
  - HighOn events
  - Private phenomena:
  - Labels
  - Linked
  - Location
  - LocationOn
  - xCenter
  - ySides

Shared Phenomena and (Ω)
- The shared phenomena are in the (small) intersection between two sets of phenomena:
  - PW
  - PM

Modelling and Shared Phenomena
- Shared phenomena and modelling are different relationships between the Machine and the World
  - Shared phenomena → modelling
  - Any description that is true of the shared phenomena is a shared description
  - But...
  - ... (modelling → shared phenomena)
  - The database shares no phenomena with the reality it models
Engineering: Requirements, Specifications, and Programs

- The purpose of the machine is to change the world; this is the requirement.
- The required changes are expressible entirely in terms of phenomena of the world...
- ...but not usually entirely in terms of phenomena shared with the machine.
- The final engineering product:
  - Machine behavior according to the program...
  - ...thus satisfying the specification and...
  - ...thus ensuring achievement of the requirement.

Requirements, Specifications, Programs

- A specification is also a requirement.
- A specification is also a program.

Engineering and Ø

- Programs can satisfy specifications only by virtue of properties of the machine (Ø semantics).
- Specifications can satisfy requirements only by virtue of properties of the world.
- The engineering is in determining, describing and exploiting the properties of the world.

A Little Engineering Example

- R: on_runway <-> can_reverse
- D1: wheel_pulse <-> wheel_turning
- D2: wheel_turning <-> on_runway
- S: can_reverse <-> wheel_pulse
- We have: S, D1, D2 -> R — is it enough?

Properties of the World

- Solution structure should reflect problem structure.
  - There’s a lean need for invention.
  - It’s easier to validate the solution.
- Traditional solution structures are often hierarchical and homogeneous.
  - Procedure hierarchies, class hierarchies, layered abstract machines, process/flowchart structures.
- ...but the World rarely exhibits such structures.

The Problem Facet of the Relationship
A Simple Editing Tool

- Three requirements:
  - Editing allows users to create and edit texts
  - GUI provides convenient and efficient operation
  - Revision History provides progress reporting by users and texts
- The requirements are related by conjunction:
  - Editing & GUI & Revision History
- The requirements share phenomena

Two Requirements Sharing Phenomena

Problem Structures

- Problems are usually structured as subproblems that are:
  - heterogeneous
  - related by superimposition
  - pinned together at shared phenomena
- The appropriate metaphor is...
  - ... not assemblies and sub-assemblies
  - ... but CYMK separations in color printing

The World and Us (1)

"The world is too much with us..."
—William Wordsworth

4 Kinds of Denial

- How we may deny our involvement
  - Denial by Prior Knowledge
  - Denial by Hacking
  - Denial by Abstraction
  - Denial by Vagueness

Denial by Prior Knowledge

"We don't need a requirements capture phase. The problem is already well defined; our task is merely to solve it."

- Automobile designers don't have a requirements capture phase...
  - The car shall be able to travel over snowdrifts and underwater
  - The car shall be able to lift a load of 5 tons
  - The car shall accommodate 10 passengers each of weight up to 500 pounds
  - ... it would be called 'Rethinking the Motor-car'
Denial by Prior Knowledge

- Legitimate only in applications that are both 
  specialized and standardized.
- Both bridge design and automobile design are 
  specialized.
- Not only automobile design is standardized (human 
  beings, roads and baggage don't vary much).
- Bridge design is not standardized (each location 
  has unique characteristics).

Denial by Hacking

- Computers are beautiful and fascinating.
  "... Miss Byron, young as she was, understood its 
  working and saw the great beauty of the invention.
  Mrs. De Morgan, on Ada’s visit to Babbage, 1828"
- Applications are often less interesting.
  "I came into this job to work with computers, not 
  to be an amateur stockbroker."
  Member of failed development team, 1993
- The Machine in the developers' own creations; 
  the World is not.

Looking at the Problem Context

- Which is the World? Which is the Machine?
- Which do you describe at the next level of DFDs?

Denial by Abstraction

"We come now to the decisive step of mathematical 
abstraction: we forget what the symbols stand for."
Norman Ryle, quoted by Abelson & Sacerdoti
- Abstraction is a valuable intellectual tool ...
- ... but it must not be a rule of life for software 
  developers.
- Too much abstraction blinds you to the nature 
  of many problems.

Doing Justice to the Problem

"One tribe always tells the truth and the other always 
lies. A traveller meets two men, and asks the first:
"Are you a truth teller?" The reply is "Goom". The 
second says: "He said Yes, but he is lying."
Marvin, Gardner, 2nd Book of Puzzles
- Abstract answer:
  "The reply must always be Yes, so the second 
  man is a truth teller, and the first is a liar."
- Lucy Jones’ answer:
  "The first man clearly can’t speak English. "Goom" 
must mean "What?" or "Welcome to our land!"
So the second man is a liar, and the first is a 
truth teller."
The Package Router

Denial by Vagueness

- Central technique:
  - Describe the Machine, but imply that you're describing the World

- Prerequisite:
  - Avoid saying explicitly what is being described

- Facilitators:
  - The modelling relationship (the same description is true of both)
  - The shared phenomena at the interface (two sides of the same penny, isn't it?)

The System and the Real World

"... the Z approach is to construct a specification document which consists of a judicious mix of informal prose with precise mathematical statements... the informal text can be consulted to find out what aspects of the real world are being described... The formal text in the other hand provides the precise definition of the system and hence can be used to resolve any ambiguities present in the informal text."

- Machine = system?
- World = real world?
- Which is being described?

Talking About the World: 4 Principles

von Neumann's principle
- Knowing what you're talking about

The principle of reductionism
- Finding the solid ground

The Shipton principle
- Recognising versatility

Montaigne's principle
- Minding your language

von Neumann's Principle

"There is no point in using exact methods where there is no clarity in the concepts and hence to which they are to be applied."

von Neumann & Margenau: Theory of Games

- Designations
- Mother(x,y) = 'x is the genetic mother of y'
- Formal term = recognition rule
- Anticipate interventions of the form:
- "It all depends on what you mean by mother?"

Aligning a Description

Orthographic Survey

Triangulation Point

- Designated terms and phenomena are like triangulation points on the map and on the ground
The Principle of Reductionism

- In any informal world many terms — often nouns in English — are obviously important...
  - in telephony: call
  - in a meeting-scheduling system: meetings
  - in an airline system: flights
  - ... but difficult or even impossible to designate
- They must be reduced to elementary designated phenomena — often events

Reducing Domain Concepts

- Designated Terms
  - take-off, land, board, disembark
- Rebuilt Terms
  - trip, stage

- The rebuilt defined terms are not the original informal terms
- Definition is not designation

The Shanley Principle

- In civil engineering design it is presently a mandatory concept known as the Shanley Design Criteria to collect several functions into one part.

1. 1940-1945 Second World War had separate components for fuel tank, outer skin, body frame
2. Samo had a tubular body of the skin at once its fuel tank, outer skin, and body frame
- It may (or may not) be good to register Machines in this way, but the World is certainly like this!
- No class hierarchy, no strong typing!

Shanley and Many Descriptions

- Editing Requirement: Operator D requested on W 7
- Revision History Requirement: Operator E requested on TW 7 by user U

- One description is not enough

Montaigne’s Principle

- “The greater part of this world’s troubles are due to questions of grammar.”
- Demanded for some Government contracts:
  - “Absolute tense ‘shall’: a binding, measurable requirement...
    - Future tense ‘will’: a reference to the future, ... not under control of the system being specified.
    - Present tense: for all other verbs ...
  - The distinction is not of causes, but of methods
    - Optative: useful in the World
    - Indicative: true regardless of the Machine

Indicative and Optative

- Natural language distinctions are impractical:
  - "I shall frame, no-one shall save me!"
  - "I will frame, no-one shall save me!"
- Mood of a sentence in development changes with its context:
  - In handling the Revision History requirement, the Editing requirement should be treated as satisfied — not optative but indicative
- So indicative and optative sentences should be kept apart in separate descriptions
Good night

- Hope you enjoyed your night at the movies with Michael Jackson
- Let’s leave discussion to the mailing list