Intro to Domain-Specific Software Engineering

Software Architecture
Lecture 23

Objectives

- Concepts
  - What is domain-specific software engineering (DSSE)
  - The "Three Lampposts" of DSSE: Domain, Business, and Technology
  - Domain Specific Software Architectures
- Product Lines
- Relationship between DSSAs, Product Lines, and Architectural Styles
- Examples of DSSE at work

Objectives

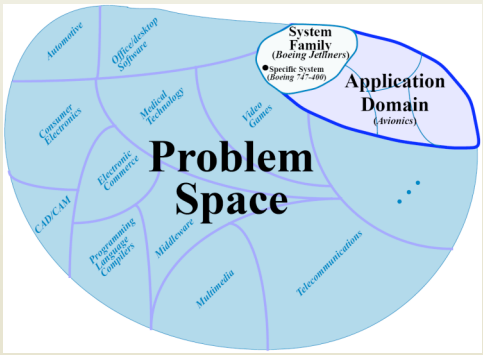
- Concepts
  - What is domain-specific software engineering (DSSE)
  - The Three Key Factors of DSSE: Domain, Business, and Technology
  - Domain Specific Software Architectures
- Product Lines
- Relationship between DSSAs, Product Lines, and Architectural Styles
- Examples of DSSE at work

Domain-Specific Software Engineering

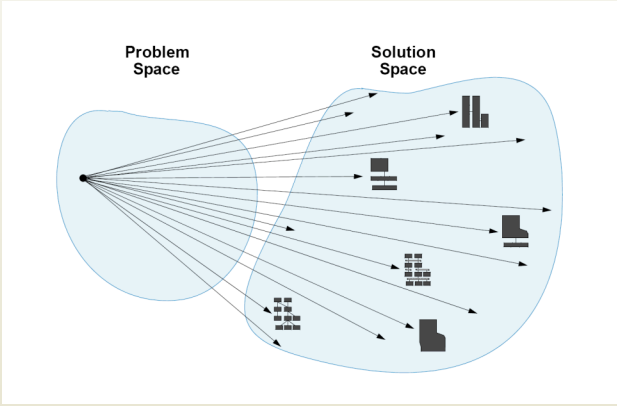
- The traditional view of software engineering shows us how to come up with solutions for problems *de novo*
- But starting from scratch every time is infeasible
  - This will involve re-inventing many wheels
- Once we have built a number of systems that do similar things, we gain critical knowledge that lets us exploit common solutions to common problems
  - In theory, we can simply build "the difference" between our new target system and systems that have come before

Examples of Domains

- Compilers for programming languages
- Consumer electronics
- Electronic commerce system/Web stores
- Video game
- Business applications
  - Basic/Standard/"Pro"
- We can subdivide, too:
  - Avionics systems
    - Boeing Jets
      - Boeing 747-400

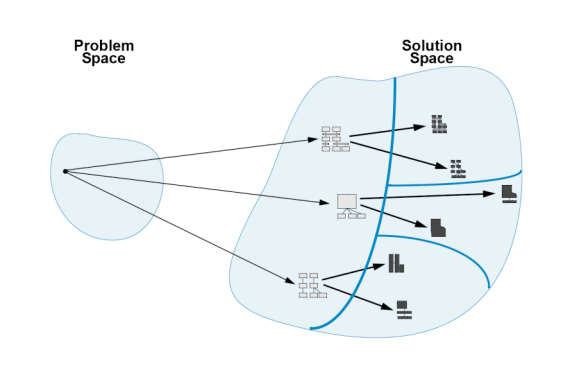


Traditional Software Engineering



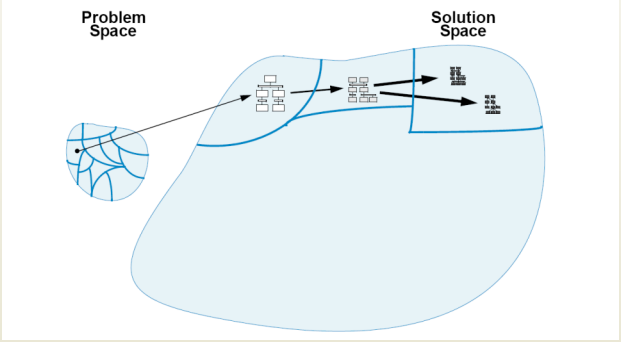
- One particular problem can be solved in innumerable ways

Architecture-Based Software Engineering



- Given a single problem, we select from a handful of potential architectural styles or architectures, and go from these into specific implementations

Domain-Specific Software Engineering



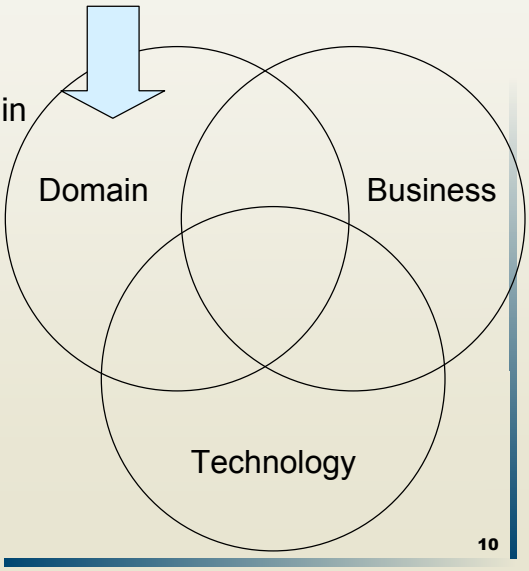
- We map regions of the problem space (domains) into domain-specific software architectures (DSSAs)
- These are specialized into application-specific architectures
- These are implemented

Three Key Factors of DSSE

- Domain
  - Must have a domain to constrain the problem space and focus development
- Technology
  - Must have a variety of technological solutions—tools, patterns, architectures & styles, legacy systems—to bring to bear on a domain
- Business
  - Business goals motivate the use of DSSE
    - Minimizing costs: reuse assets when possible
    - Maximize market: develop many related applications for different kinds of end users

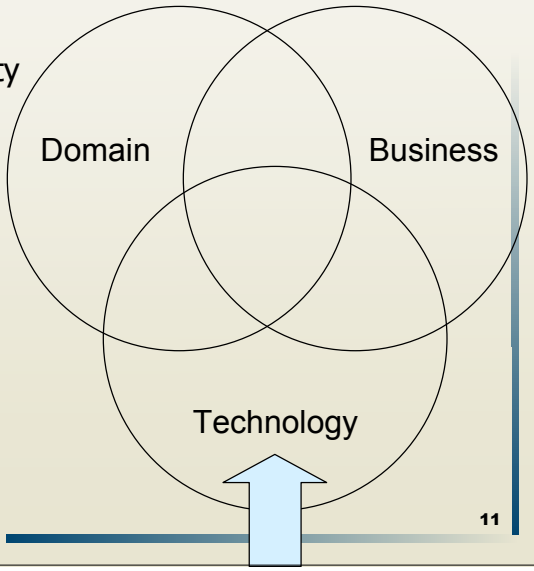
Three Key Factors

- Domain
  - Must have a domain to constrain the problem space and focus development



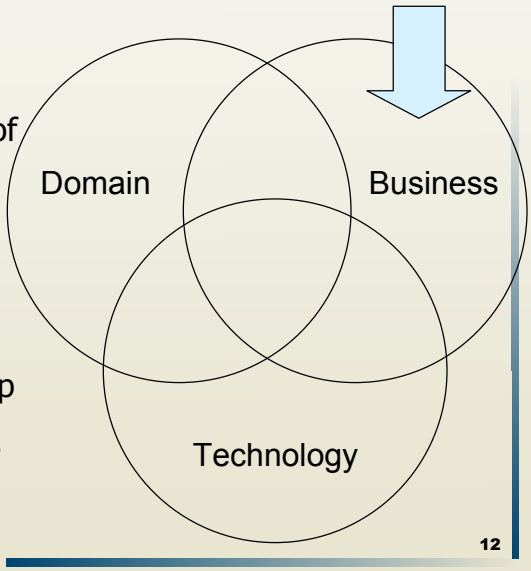
Three Key Factors

- Technology
  - Must have a variety of technological solutions—tools, patterns, architectures & styles, legacy systems—to bring to bear on a domain



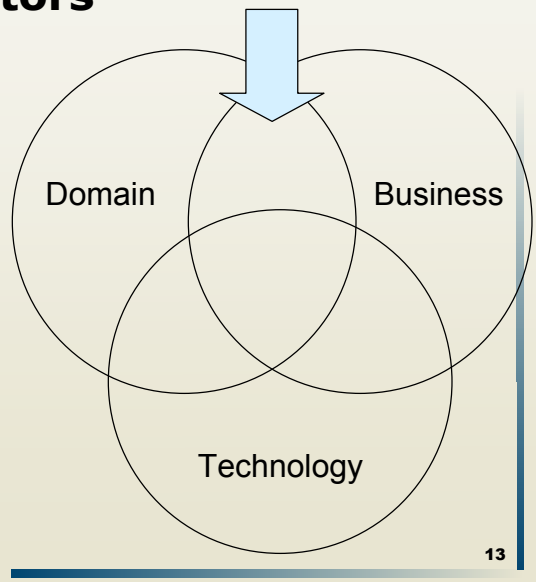
Three Key Factors

- Business
  - Business goals motivate the use of DSSE
    - Minimizing costs: reuse assets when possible
    - Maximize market: develop many related applications for different kinds of end users



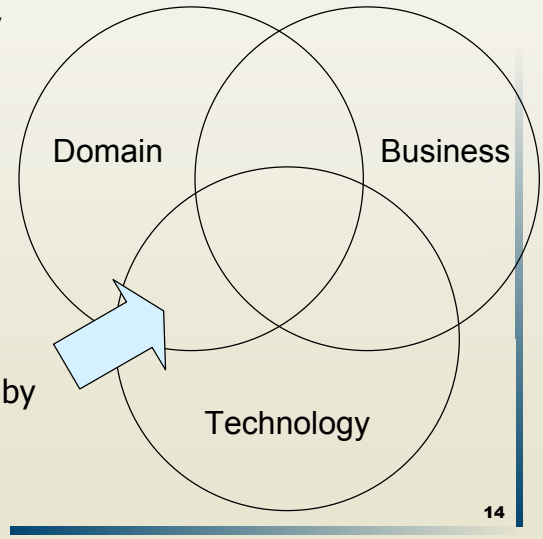
Three Key Factors

- Domain + Business
- "Corporate Core Competencies"
  - Domain expertise augmented by business acumen and knowledge of the market



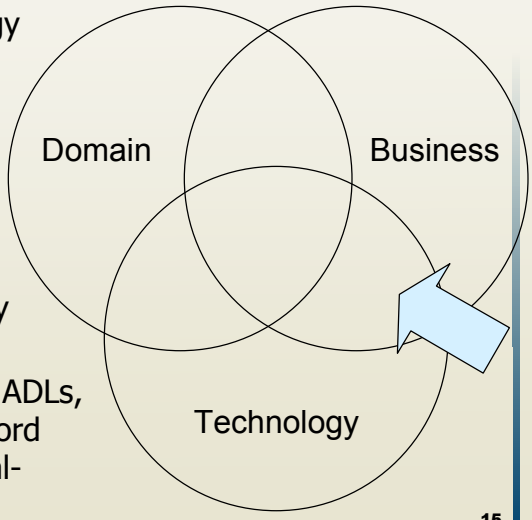
Three Key Factors

- Domain + Technology
- "Application Family Architectures"
  - All possible technological solutions to problems in a domain
  - Uninformed and *unconstrained* by business goals and knowledge



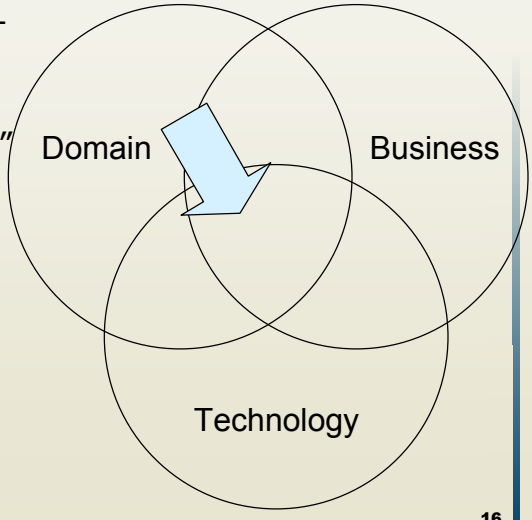
Three Key Factors

- Business + Technology
- "Domain independent infrastructure"
  - Tools and techniques for constructing systems independent of any particular domain
  - E.g., most generic ADLs, UML, compilers, word processors, general-purpose PCs



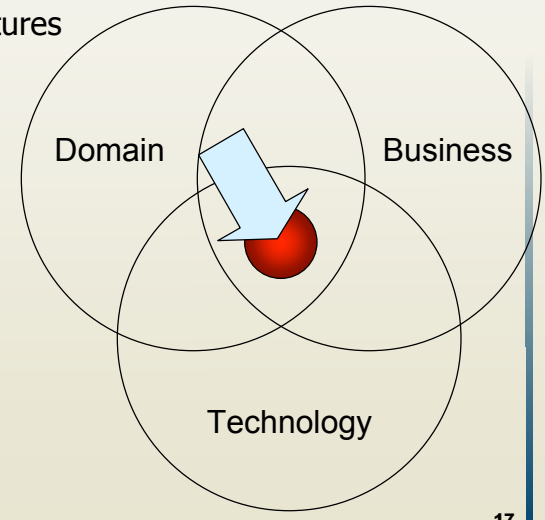
Three Key Factors

- Domain + Business + Technology
- "Domain-specific software engineering"
- Applies technology to domain-specific goals, tempered by business and market knowledge



Three Key Factors

- Product-Line Architectures
- A specific, related set of solutions within a broader DSSE
- More focus on commonalities and variability between individual solutions



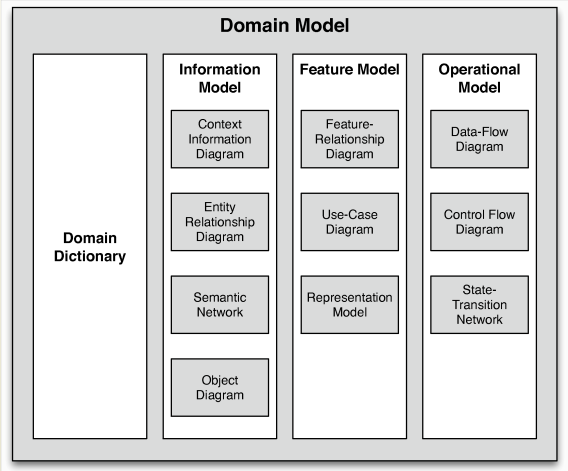
Becoming More Concrete

- Applying DSSE means developing a set of artifacts more specific than an ordinary software architecture
  - Focus on aspects of the domain
  - Focus on domain-specific solutions, techniques, and patterns
- These are
  - A *domain model* and
  - A domain-specific software architecture (DSSA)

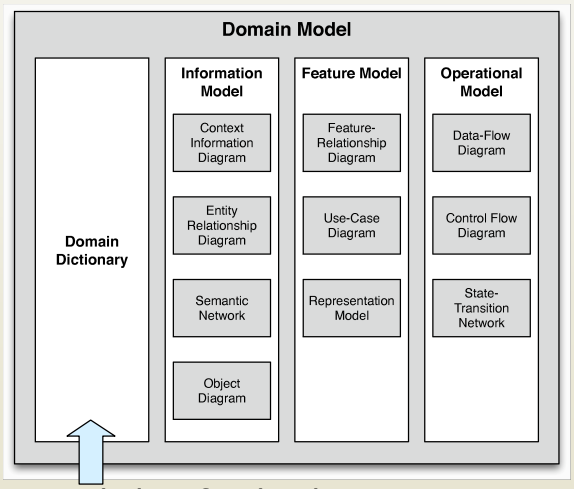
Domain Model

- A domain model is a set of artifacts that capture information about a domain
  - Functions performed
  - Objects (also known as entities) that perform the functions, and on which the functions are performed
  - Data and information that flows through the system
- Standardizes terminology and semantics
- Provides the basis for standardizing (or at least normalizing) descriptions of problems to be solved in the domain

Domain Model

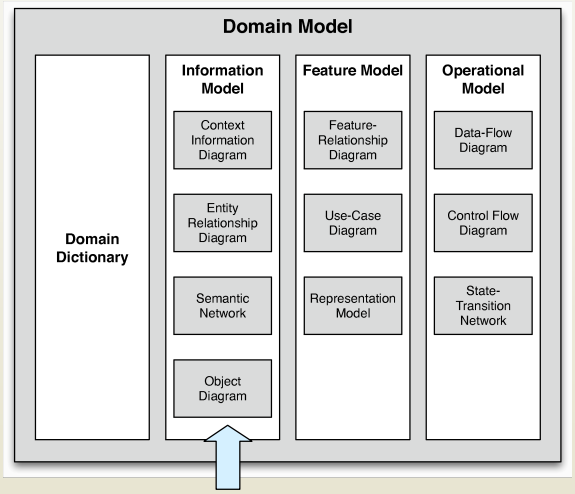


Domain Model



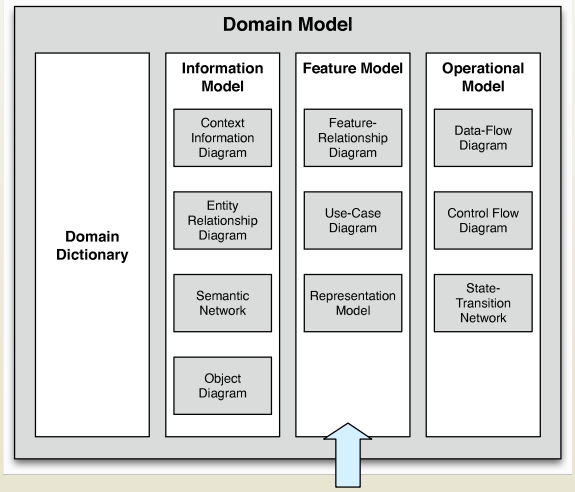
- Defines vocabulary for the domain

Domain Model



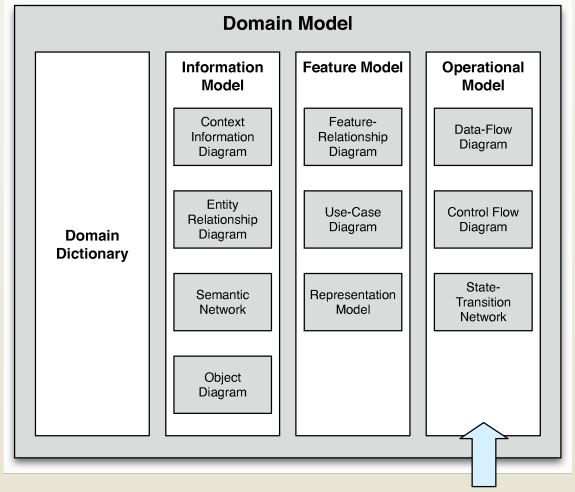
- Describes the entities and data in the domain

Domain Model



- Defines how entities and data combine to provide features

Domain Model



- Defines how data and control flow through entities

(Partial) Domain Dictionary

Lunar Module (LM): this is the portion of the spacecraft that lands on the moon. It consists of two main parts: the Ascent Stage (which holds the crew cabin) and the Descent Stage, which contains thrusters used for controlling the landing of the LM.

Reaction Control System (RCS): a system on the Lunar Module responsible for the stabilization during lunar surface ascent/descent and control of the spacecraft's orientation (attitude) and motion (translation) during maneuvers

Vertical velocity (see also One-dimensional motion):
For a free-falling object with no air resistance, ignoring the rotation of the lunar surface, the altitude is calculated as follows:

$$y = \tfrac{1}{2} * a * t_2 + v_i * t + y_i$$

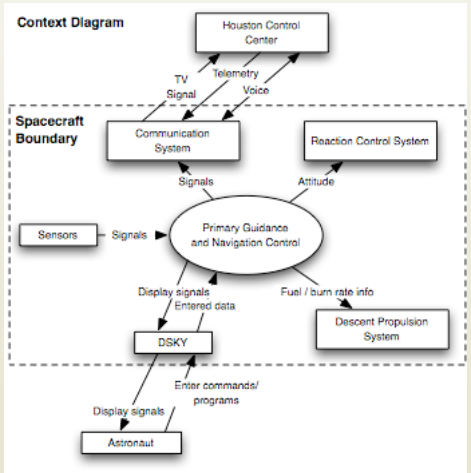
y = altitude
a = constant acceleration due to gravity on a lunar body (see **Acceleration** for sample values)
t = time in seconds; v_i = initial velocity; y_i = initial altitude
When thrust is applied, the following equation is used:

$$y = \tfrac{1}{2} * (a_{burner} - a_{gravity}) * t_2 + v_i * t + y_i$$

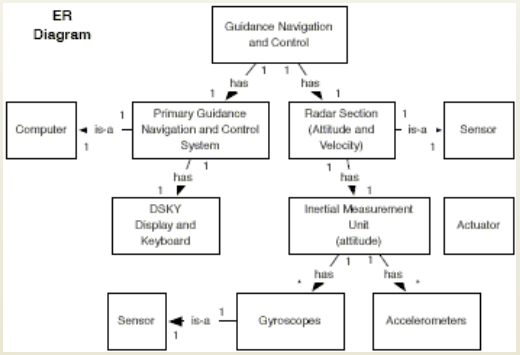
y = altitude
a_{burner} = constant acceleration upward due to thrust
$a_{gravity}$ = constant acceleration due to gravity on a lunar (see **Acceleration** for sample values)
t = time in seconds; v_i = initial velocity; y_i = initial altitude

Info Model: Context Info Diagram

- Defines high-level entities
- Defines what is considered inside and outside the domain (or subdomains)
- Defines relationships and high-level flows



Info Model: Entity-Relationship Diagram



- Defines entities and cardinal relationships between them

Info Model: Object Diagram

- Defines attributes and operations on entities
- Closely resembles class diagram in UML but may be more abstract

Object	Attributes	Operations
Landing Radar	Altitude Velocity	Sense Altitude (height) Sense Velocity
Descent Engine	Throttle Level Nozzle Direction	Set Throttle Level Change Nozzle Direction (Gimbal) Turn On / Off
Thrust Engines	Firing Mode Attitude Translation	Set Firing Mode to pulse or continuous Change Spacecraft Attitude Change Spacecraft Translation

Feature Model: Feature Relationship Diagram

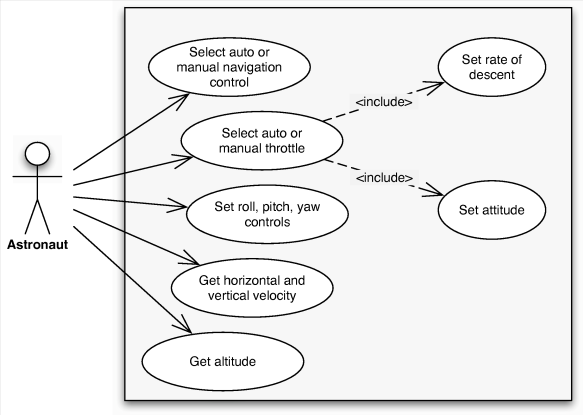
Feature Relationship Diagram – Landing Phase

Mandatory: The Lunar Lander must continually read altitude from the Landing Radar and relay that data to Houston with less than 500 msec of latency. Astronauts must be able to control the descent of the Lunar Lander using manual control on the descent engine. The descent engine must respond to control commands in 250msec, with or without a functioning DSKY…
Optional/Variant: Lunar Lander provides the option to land automatically or allow the crew to manually steer the spacecraft.
Quality Requirements:
Real-time requirements: The thrusters and the descent engine must be able to respond to commands from the computer system in real-time.
Fault tolerance: Lunar Lander must be able to continue in its flight-path even when the main computer system (Primary Navigation Guidance & Control) goes down. Lunar Lander must be able to maintain system altitude even when one of the thrusters and propellant supplies goes down in the Reaction Control System.

- Describes overall mission operations of a system
- Describes major features and decomposition

Feature Model: Use Case Diagram

- Defines use cases within the domain
- Similar to use case models in UML



Feature Model: Representation Diagram

- Defines how information is presented to human users

Representation Diagram

DSKY Unit – Display and Keyboard Unit

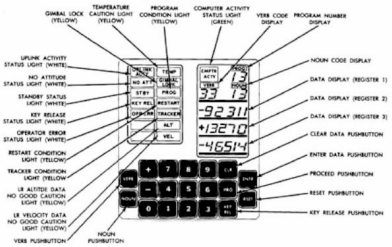


Figure 3: Lunar Module Display and Keyboard Unit (DSKY)

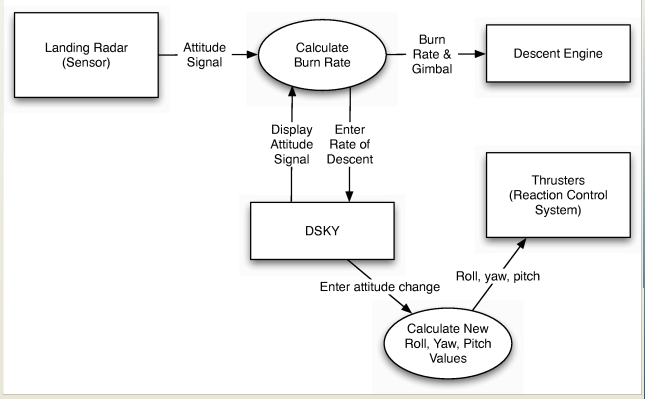
Source: TALES FROM THE LUNAR MODULE GUIDANCE COMPUTER – figure
Apollo 11 The NASA Mission Reports Vol 2 pp 166

- 3 five-digit registers – general purpose
- 3 two-digit registers – indicate phase for landing
- 19 keys
- Warning lights
- Issue commands via VERB & NOUN
  - VERB is the action
  - NOUN is the object to which the action is applied
  - Ex: VERB 6 NOUN 20
    VERB 6 = Display in decimal
    NOUN 20 = Angles
- 70 predefined PROGRAMS
  - Ex: PROGRAM for each descent phase executes trajectory
    - P63 – Braking Phase
    - P64 – Approach Phase
    - P65 – Landing Phase

Operational Model: Data Flow Diagram

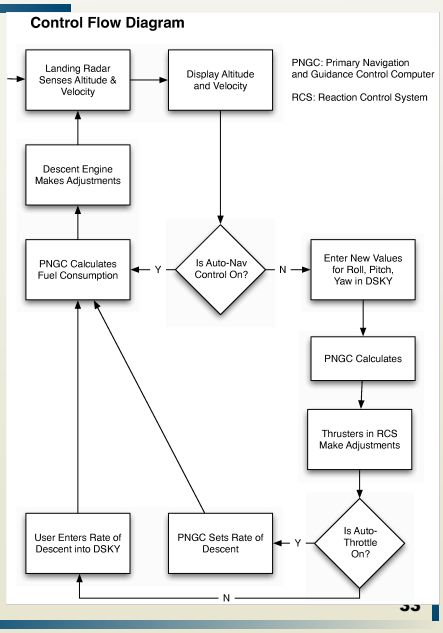
- Focuses on data flow between entities with no notion of control

Data Flow Diagram



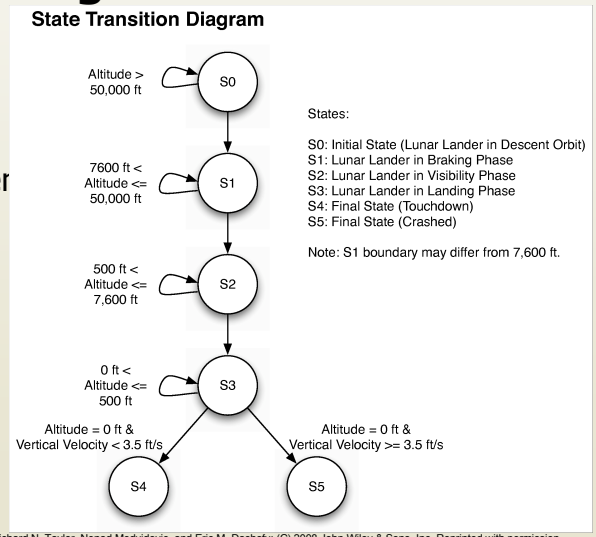
Operational Model: Control Flow Diagram

- Focuses on control flow between entities separate from data flow



Operational Model: State Transition Diagram

- Focuses on states of systems and transitions between them
- Resembles UML state diagrams



Reference Requirements

- ***Mandatory***
  - *Must display the current status of the Lunar Lander (horizontal and vertical velocities, altitude, remaining fuel)*
  - *Must indicate points earned by player based on quality of landing*
- ***Optional***
  - *May display time elapsed*
- ***Variant***
  - *May have different levels of difficulty based on pilot experience (novice, expert, etc)*
  - *May have different types of input depending on whether*
    - *Auto Navigation is enabled*
    - *Auto Throttle is enabled*
  - *May have to land on different celestial bodies*
    - *Moon*
    - *Mars*
    - *Jupiter's moons*
    - *Asteroid*

Domain-Specific Software Architecture

- **Definition:** Definition. A domain-specific software architecture (DSSA) comprises:
  - a reference architecture, which describes a general computational framework for a significant domain of applications;
  - a component library, which contains reusable chunks of domain expertise; and
  - an application configuration method for selecting and configuring components within the architecture to meet particular application requirements.

(Hayes-Roth)

Reference Architecture

- **Definition.** *Reference architecture* is the set of principal design decisions that are simultaneously applicable to multiple related systems, typically within an application domain, with explicitly defined points of variation.
- Reference architectures are still architectures (since they are also sets of principal design decisions)
  - Distinguished by the presence of explicit points of variation (explicitly "unmade" decisions)

Different Kinds of Reference Architecture

- Complete single product architecture
  - A fully worked out exemplar of a system in a domain, with optional documentation as to how to diversify
    - Can be relatively weak due to lack of explicit guidance and possibility that example is a 'toy'
- Incomplete invariant architecture
  - Points of commonality as in ordinary architecture, points of variation are indicated but omitted
- Invariant architecture with explicit variation
  - Points of commonality as in ordinary architecture, specific variations indicated and enumerated

Example Reference Architecture

- Structural view of Lunar Lander DSSA
- Invariant with explicit points of variation
  - Satellite relay
  - Sensors