The World and the Machine

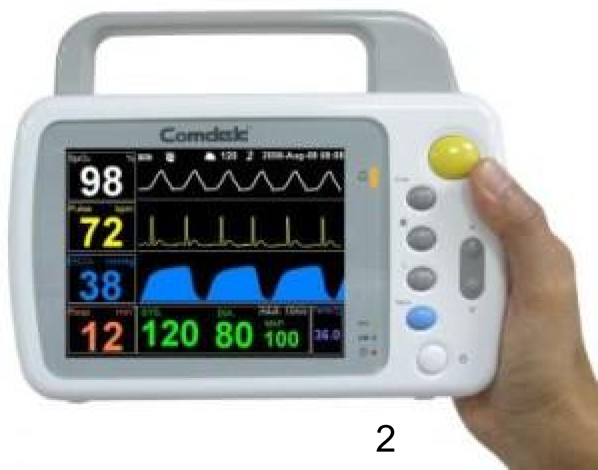
CSCE 740 - Lecture 9 - 09/23/2015

Patient Monitor

- Requirement: A nurse must be notified if the patient's heart stops.
- Specification states:
  - When the sensed heartbeat falls below a defined threshold, the alarm shall be actuated.

Will this work?

(assume the thresholds are defined)



How do we know that the software will work?

(AKA: How do we know that our specification is correct?)

Requirements and Specifications

- The requirements are things that we **want** the software to make true.
- We write specification statements describing what the software **will do** to make the requirements true.

The World and the Machine

- Software is a description of a machine.
- But, the purpose of the machine is located in the world in which it is installed and used.
- The needs of the user - the problems we are solving - are part of the **world domain**, and the solution we construct forms the **machine domain**.
- The specification must bridge the two domains.

How Do We Know the Software Will Work?

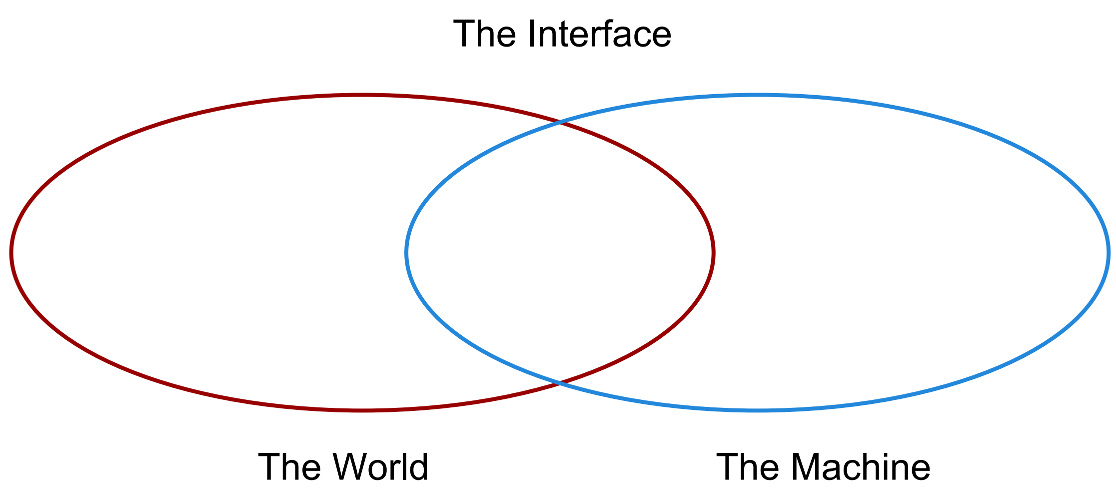
- Understand the requirements.
- Clearly state our domain knowledge and assumptions about the world that the software will operate within.
  - **This is critical!**
- Write specifications that - as long as our assumptions hold - will enable the machine to satisfy the real-world requirements.

Objectives For Today

- Introduce the World-Machine Model
  - A framework for analyzing relationships between the requirements, specification, machine, and domain assumptions.
- Understand the relationships between the machine and the world.
- Understand how to capture those relationships in the system specification.
- Discuss common mistakes made during system specification.

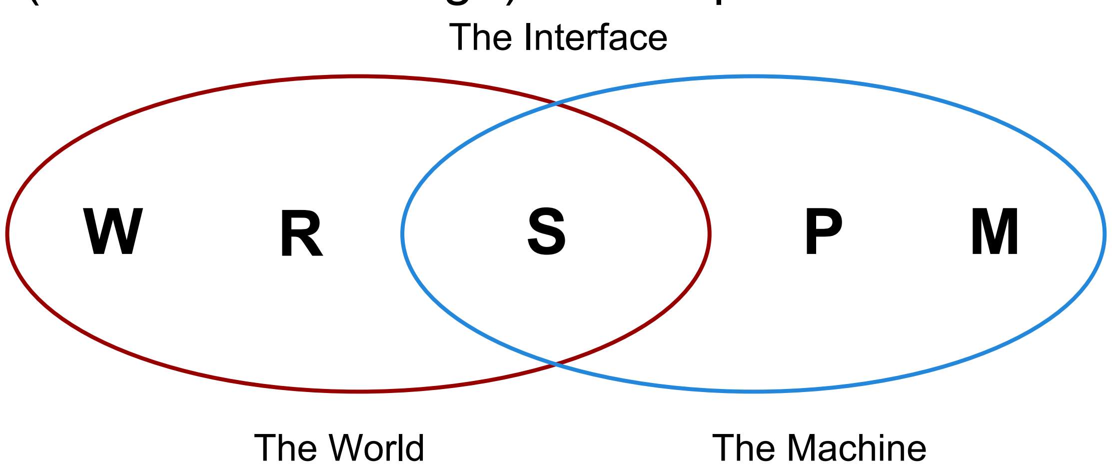
The World-Machine Model

- We want to make a change to the environment (world).
- We will build a machine to do it.
- The machine must interact with the world.

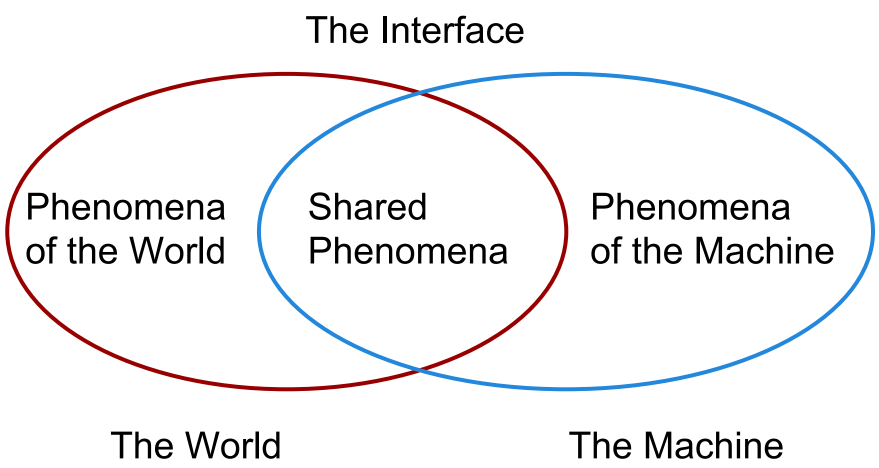


The World-Machine Model

- R (Requirements): Phenomena that we should make true.
- P (Program): The description of the machine.
- S (Specification): What the software can do to connect the machine and the world.
- M (Hardware): The physical hardware.
- W (Domain Knowledge): Assumptions about the world.



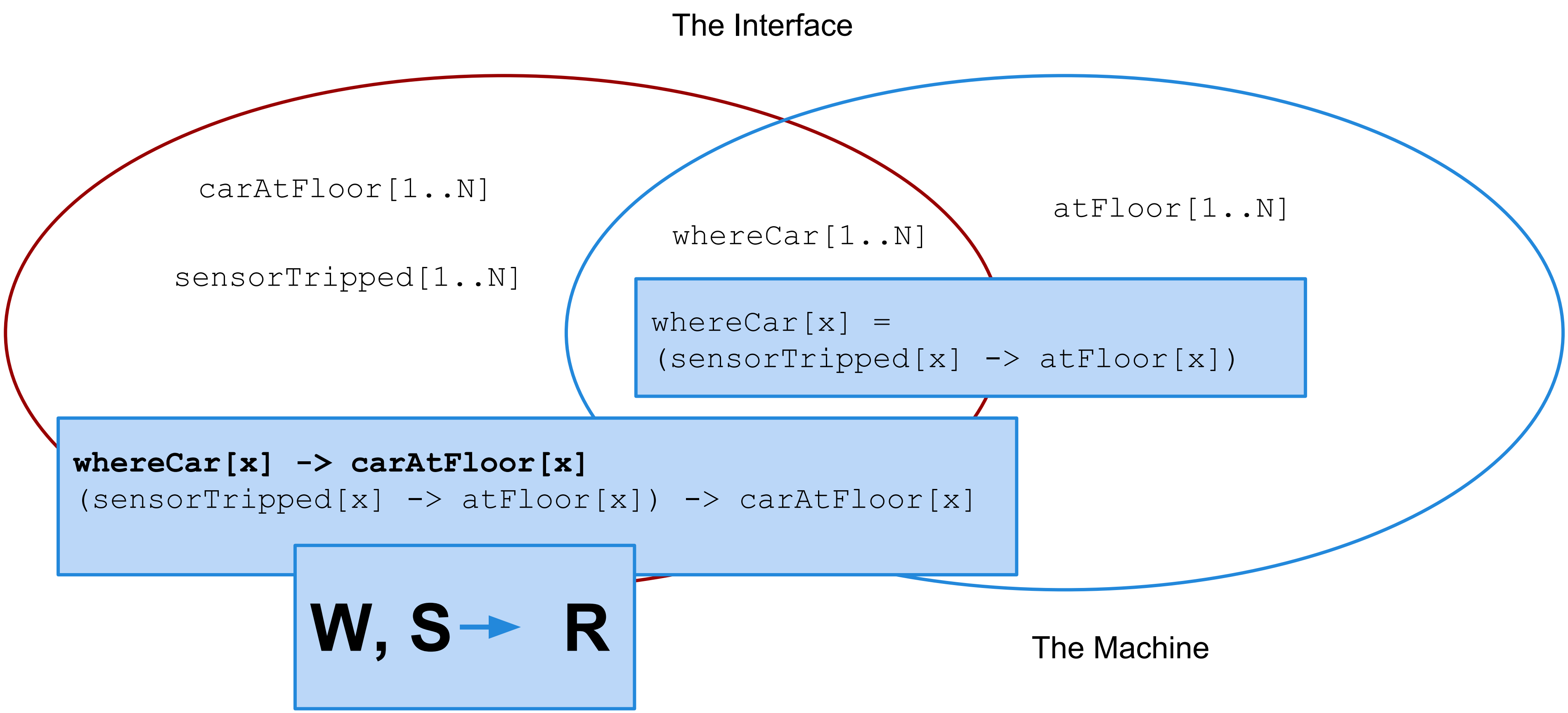
Phenomena - What We Know



We can use these variables to make logical arguments.

- Those artifacts describe phenomena of the world, machine, and interface.
  - Variables assigned to values
    - Factors we can control or measure.
  - and logical statements
    - Things we assert as fact.

Elevator System



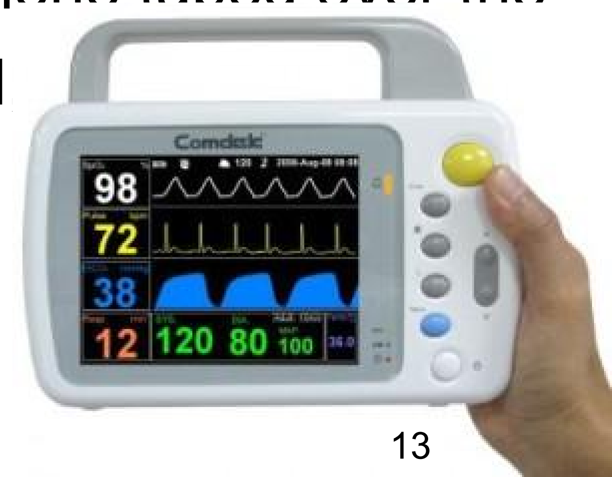
The Role of Specification

- Requirements are concerned with phenomena in the world. Programs are concerned with phenomena in the machine.
- Specifications form the gap, and are concerned with shared phenomena.
- Specification is needed because it is a staging post for implementation. Engineering and assumptions of the world are captured by the specification.

Patient Monitor

- Problem:
  - We desire a warning system that notifies a nurse if the patient’s heart stops.
- User Requirement:
  - When the patient’s heart stops, a nurse shall be notified.
- System Specification:
  - When the sensed heartbeat (microphone taped over the heart) falls below a defined threshold

Does this specification satisfy the requirement?



Patient Monitor Variables

- User Requirement:
  - When the patient's heart stops, a nurse shall be notified.
- Hardware Design:
  - A computer that can be programmed to use a microphone as a sensor and a buzzer as an actuator.
- Specification:
  - If the sound from the sensor falls below a defined threshold, the buzzer shall be actuated.

Will Patient Monitoring Work?

- If we take a computer that can be programmed to use a microphone as a sensor and a buzzer as an actuator…
- and if we program the computer to sound the buzzer when the sound from the sensor falls below a certain threshold…
- Then we will have a warning system that notifies the nurse if the patient's heart stops.

Do we believe this?

Patient Monitoring Will Work

- If we take a computer that can be programmed to use a microphone as a sensor and a buzzer as an actuator…
- and if we program the computer to sound the buzzer when the sound from the sensor falls below a certain threshold…
- Then we will have a warning system that notifies the nurse if the patient's heart stops.

Because…

- **There will always be a nurse close enough to hear the buzzer**
- **The sound from a heart falling below a certain threshold indicates that there is a heart problem.**

Requirements Satisfaction

- Domain knowledge is a set of properties of the world that we assume to be true.
- Requirements are properties of the world we *want to make true*.
- The machine is not solely responsible for satisfying the requirements. It can only function in the world it was specified for.
- We want to prove that the combination of domain knowledge (W) and specification (S) satisfies the requirements (R).

$$\boxed{W, S \rightarrow R}$$

Example - Traffic Light

Requirement (R):
Allow pedestrians to cross the road safely.

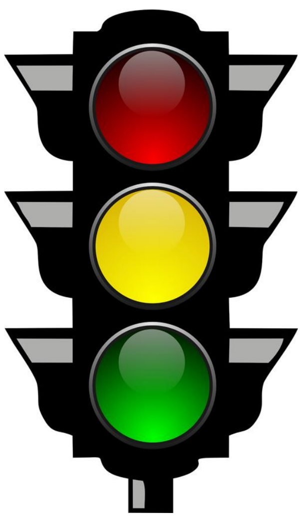
Specification (S):
Show a red light to the cars and a green light to the pedestrians.

Domain Knowledge (W):
- Drivers stop at red lights.
- Pedestrians walk when green.

$W, S \rightarrow R$

Safety Example - Traffic Light

Safety Requirement (R):
Pedestrians and cars cannot be in the intersection at the same time.



Specification (S):
When a green light is shown to a car, a red light must be shown to the pedestrian.
(and vice-versa)

Domain Knowledge (W):
- Drivers stop at red lights.
- Pedestrians stop at red lights.
- Drivers drive at green lights.
- Pedestrians walk when green.

W, S → R

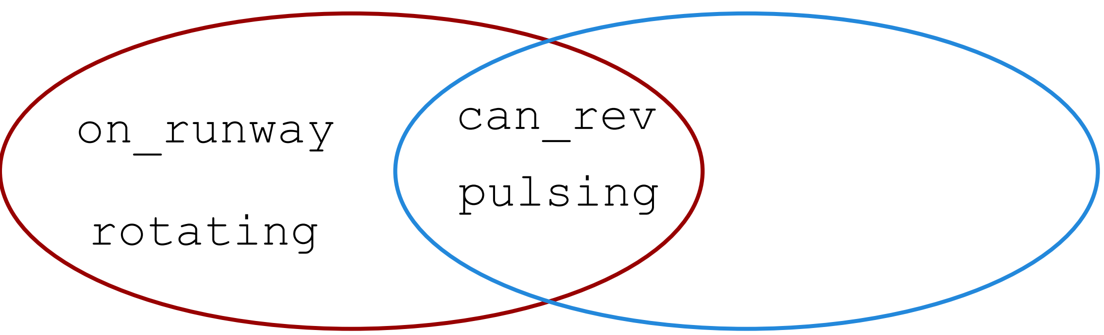
Domain Knowledge is Essential

- Driver or pedestrian behavior is not guaranteed. However, by being explicit about assumptions, we can write better specifications and plan for failures.
- The is the most error-prone part of the requirements and specification. Most problems can be traced to erroneous assumptions about the world.
- Always validate and continually question your domain knowledge.

Example: Plane Landing

- Requirement: Reverse thrust can only be engaged if the plane is on the runway.
  - `can_rev` **->** `on_runway`

The Interface



The World

The Machine

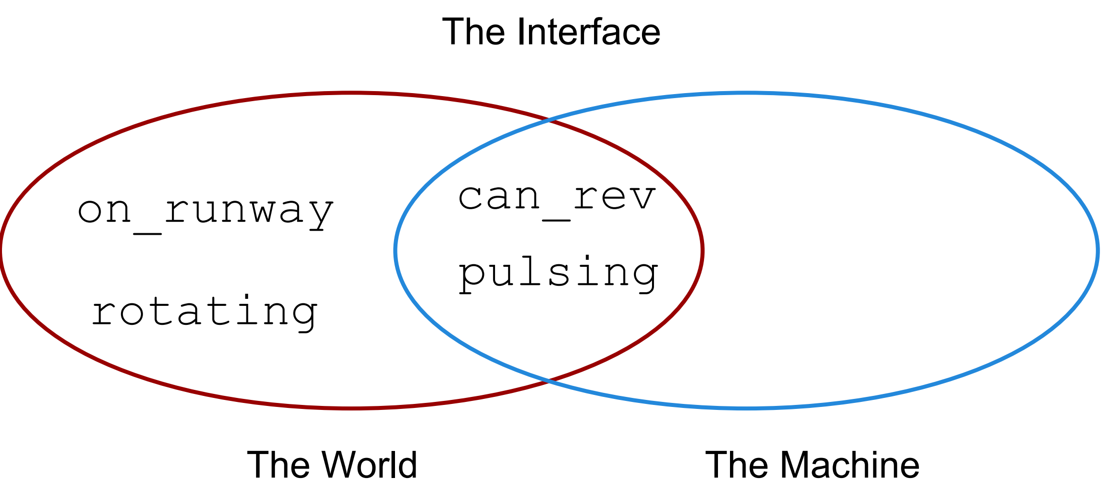
- Ability to reverse thrust is controlled by machine and reacted to by world.
- The plane being on the runway is part of the world.
- The wheels rotate in the world...
- and trigger sensors that are reacted to by the machine.

Specification:

`pulsing` **->** `can_rev`

Example: Plane Landing

- Can we argue that this specification and world satisfy `(can_rev -> on_runway)`?



Specification:

- `pulsing -> can_rev`

Domain Assumptions:

- `pulsing -> rotating`
- `rotating -> on_runway`

We argue that the combination of domain and specification satisfy the user requirement.

Example: Plane Landing

- **Problem: What if the assumptions don't hold?**
- **Specification:**
  - `pulsing` **->** `can_rev`
- **Domain Assumptions:**
  - `pulsing` **->** `rotating`
  - `rotating` **->** `on_runway`

Engineering of software can't be done in isolation. You need to understand and manipulate the world that the system operates in.

Domain Knowledge is Essential

Even if domain properties were captured correctly and the system initially met requirements, the world can change!

- Ariane 5 rocket
- New York subway system

- **ALWAYS validate and continually question your domain knowledge.**

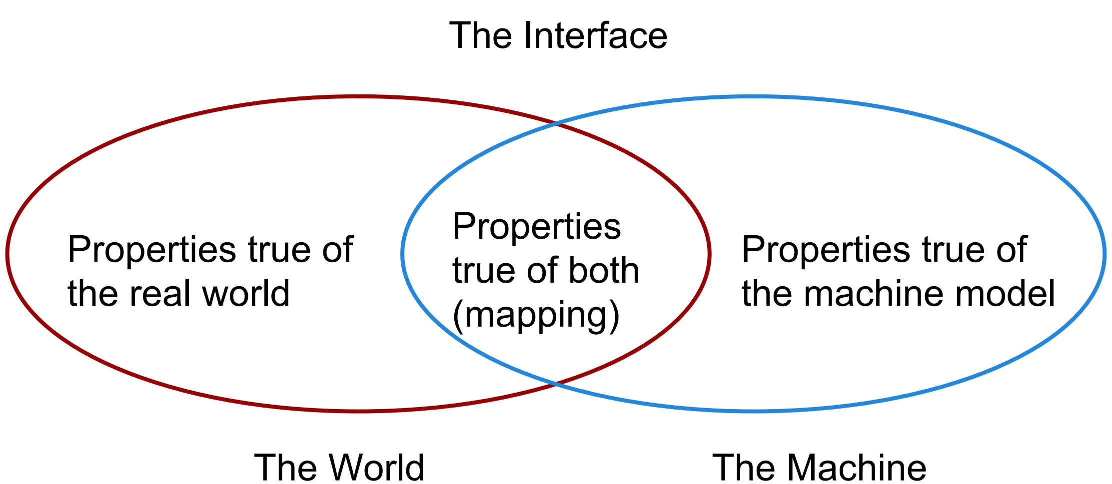
Modeling the World

There are many systems that do not directly interface with the world, but most systems do *model* a process in the world.

- The purpose of such software is to provide efficient and convenient access to information about the world.
- There is no direct interaction, but there is a *conceptual* connection between the world and the machine that must be captured.

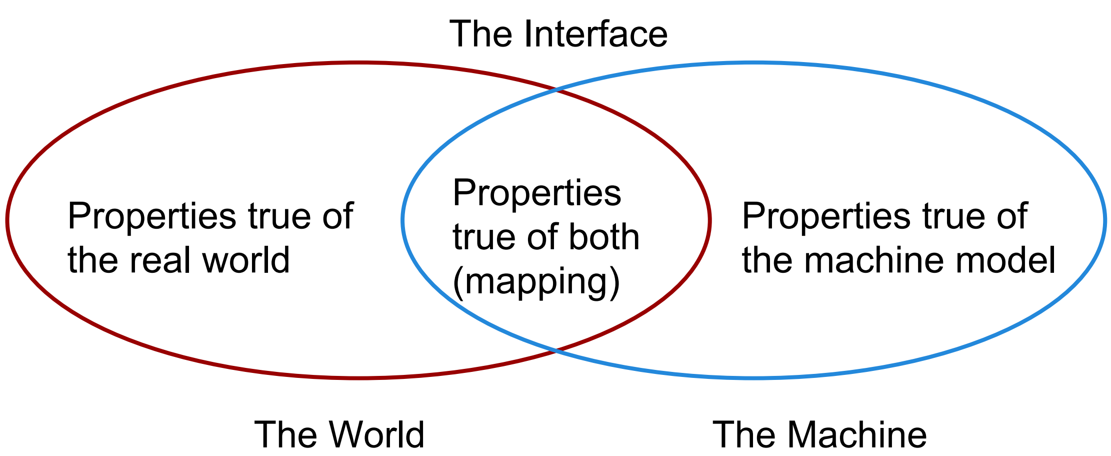
Modeling a Process

- The modeling works because there are properties true of both the world and the machine model.
- These must be described differently in the world and machine, and the specification needs to map the two forms.



Library Model

- **World Property:** For each novel X, there is a unique writer Y who is the author of the novel.
- **Machine Property:** For each record of type B, there is a unique record of type W to which there is a pointer from the B record.
- **Mapping Property:** Each B record contains a character string that is the title of the novel and each A record contains a character string that is the name of the author.



We Want to Show...

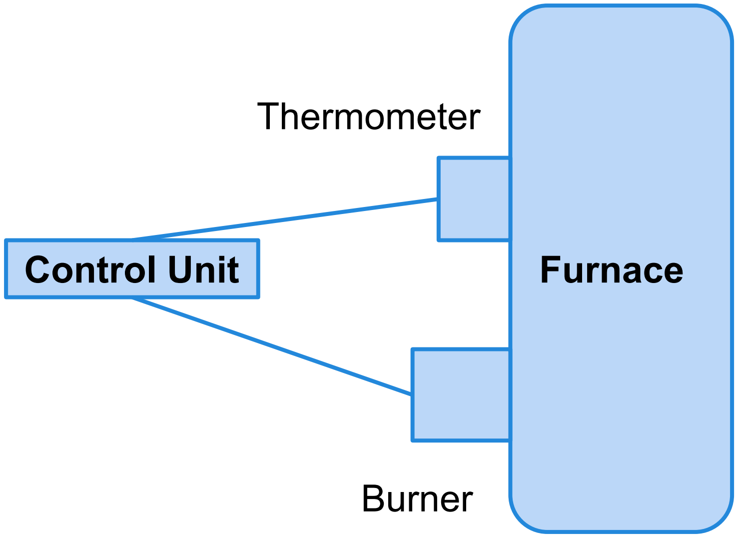
To verify the software, we need to argue that we built the software correctly. We can use the world-machine model to argue:

- W, S -> R
  - The specification satisfies the user’s requirements, under the stated domain properties.
- P, M -> S
  - The Machine satisfies the specification (verification).
- W, P, M -> R
  - The Machine satisfied the user’s requirements, under the stated domain properties.

Furnace System

We are building a system to control the temperature of a furnace.

- What forms the world and the machine?
- What are some of the variables that we can establish?

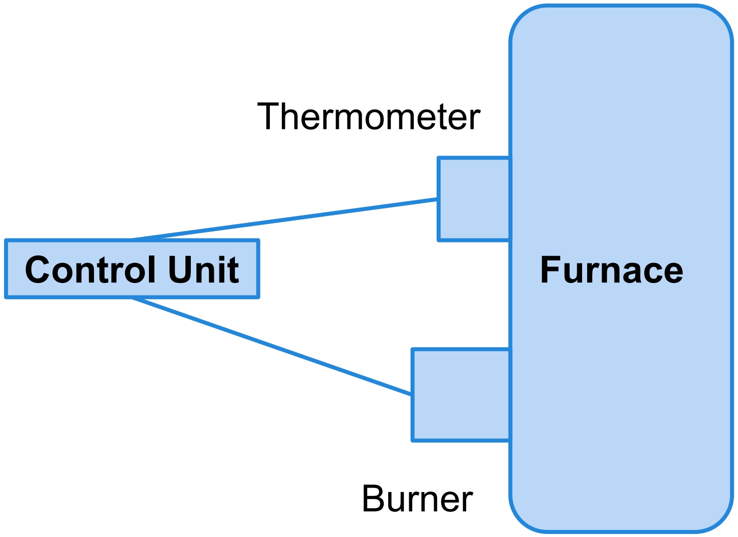


Activity - Furnace System

Requirement:

- If the furnace temperature drops below the threshold, then the burner shall be activated.

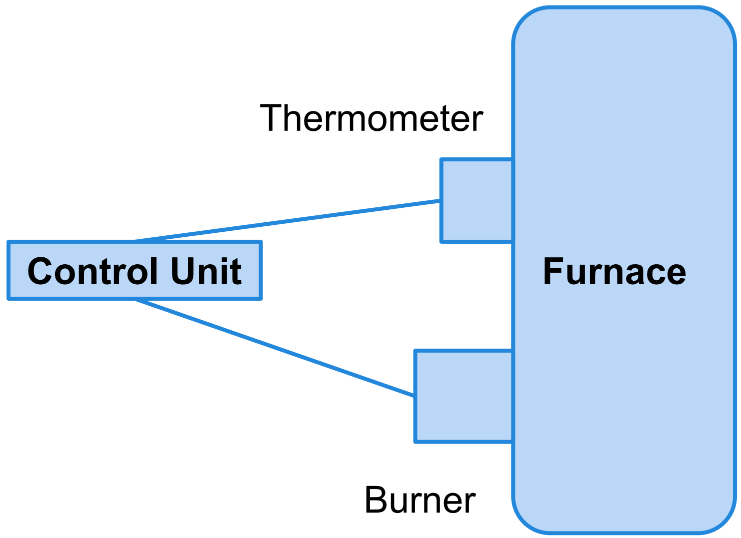
- Establish the properties of the world, machine, and interface required to satisfy this requirement.
- State the specification and domain knowledge that should satisfy the requirement.

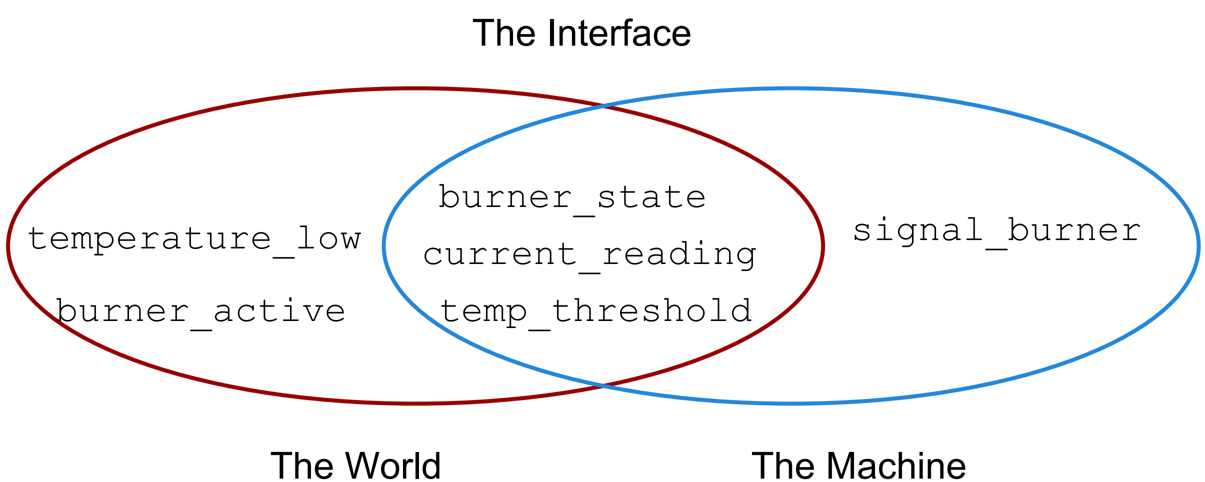


Furnace System Solution

Requirement:

- If the furnace temperature drops below the threshold, then the burner shall be activated.
- **`temperature_low -> burner_active`**





Furnace System Solution

Requirement:

- **`temperature_low` -> `burner_active`**

Specification:

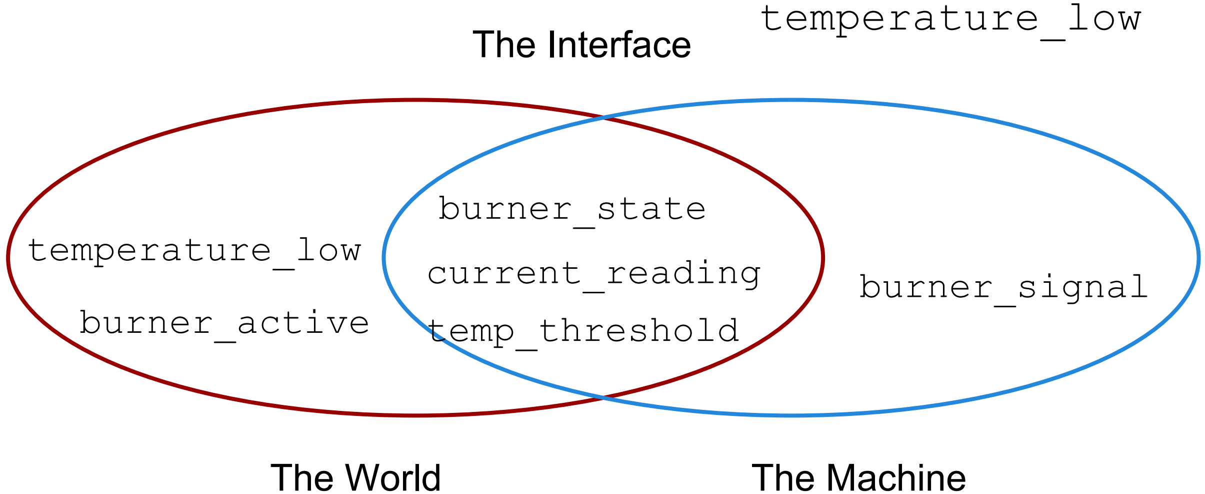
- `burner_signal` -> (`burner_state = on`)

Machine Assumptions:

- `burner_signal` -> (`current_reading` < `temp_threshold`)

Domain Assumptions:

- (`current_reading` < `temp_threshold`) -> `temperature_low`



Four Principles for Description

We need to clearly capture the world. We can do so by following four principles:

- Von Neumann’s principle
- The principle of reductionism
- The Shanley principle
- Montaigne’s principle

Von Neumann’s Principle

“There is no point in using exact methods when there is no clarity in the concepts and issues to which they are to be applied.”

- Before capturing properties of the world and machine, start by establishing the vocabulary of ground terms that we will use.
- If we want to assert that “For each novel X, there is a unique writer Y that is the author of the novel.”
    - What is a novel?
    - What is a writer?
    - What does it mean to be an author?

Von Neumann’s Principle

- For each phenomenon, we must give a:
  - **Recognition rule:** a definition of the phenomenon.
  - **Formal term:** symbol and argument list by which we can refer to the phenomenon.
- This is possible because of two bounds:
  - We are not required to formalize the whole world - only the phenomena of interest.
  - We must only formalize what is needed to meet the user’s needs.
- Formal semantics make specifications provable and refutable.

Von Neumann’s Principle

“There is no sense in being precise if you don’t know what you are talking about.”

The Principle of Reductionism

- We have freedom in choosing how we will describe phenomena.
- When possible, we should reduce the problem to the simplest building blocks, then construct more complex phenomena and properties from those blocks.
- Reduce a problem into its components, define those precisely, then use those stable components to make complex arguments.

The Principle of Reductionism

- Avoid nouns to describe phenomena. These are almost always wrong.
  - Member, Call, Meeting, Flight
- Describe nouns using the phenomena that define the noun.
  - A Member should be defined in terms of events: enrolled, resigned, lapsed, expelled.
- Ground terms should almost always be events. Use events to give meaning to a member, call, or flight.

Shanley's Principle

- Separation of concerns: hide aspects of a problem currently not of interest.
- Shanley's Principle: Separate aspects of a problem into parallel components, and do not lose site of the connections between these components.
- The world is complicated, and restricting our view of it too much can risk not capturing enough detail.

Montaigne's Principle

"The greater part of the world's troubles are due to questions of grammar."

- Need to make a clear distinction between the operative mood - *what we want to be true* - and the indicative mood - *what we assert to be true*.
- Requirements can be operative (we want the machine to make this happen), but domain knowledge must be indicative:
  - `can_rev` **->** `on_runway`
  - `rotating` **->** `pulsing`

Montaigne’s Principle

- In natural language, be careful with verbs such as “will” and “shall.”
  - Will is indicative - we assert this to be true.
  - Shall is operative - this could be made true.

- Keep in mind the distinction between:
  - “I shall drown. No one will save me.”
  - “I will drown. No one shall save me.”

We Have Learned

- Specifications must capture the relationship between the system and the world it will live in.
- We can model this relationship with the world-machine model. Capture properties of the machine and world, and properties of the interface between them.
- Always state and validate your assumptions about the world.

Next Time

- Requirements Modeling & Verification

- Readings:
  - Steven Miller - “Proving the Shalls” (on Moodle)

- Homework:
  - Homework tonight.
  - Any questions?