Identifying the Problem Space

17-423/723 Designing Large-Scale Software Systems

Recitation 1
Jan 17, 2025

Recap: Problem vs. Solution Space

Problem vs. Solution Space

Problem space (aka **domain** or **world**)

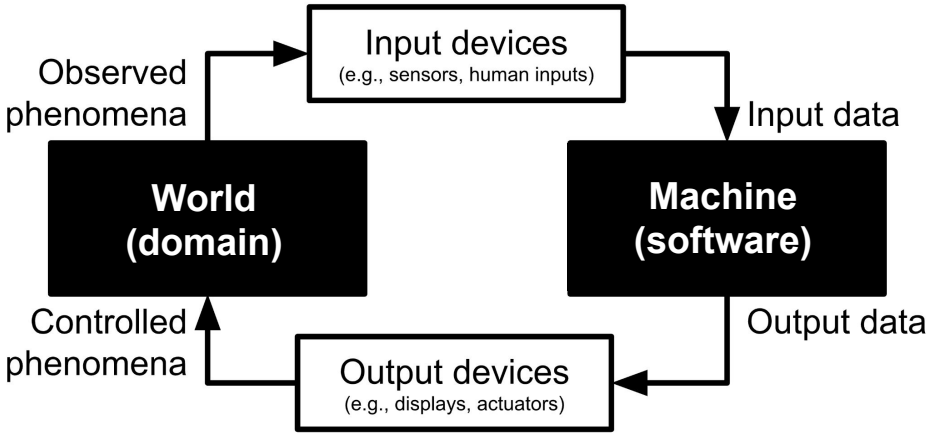
Physical entities in the real world, their behaviors & relationships

Part of the world that software may influence, but cannot directly control

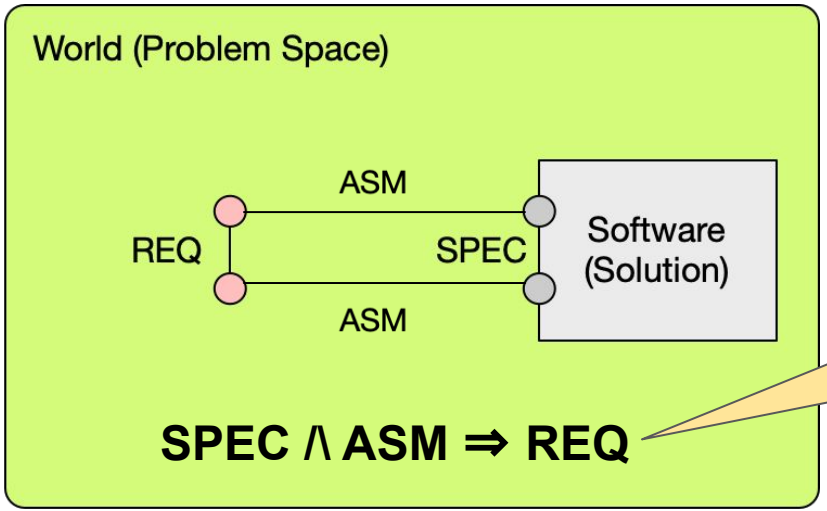
Solution space (aka **machine**)

A product (i.e., software) to be developed to solve the customers' problem

A combination of software components that you have creative control over



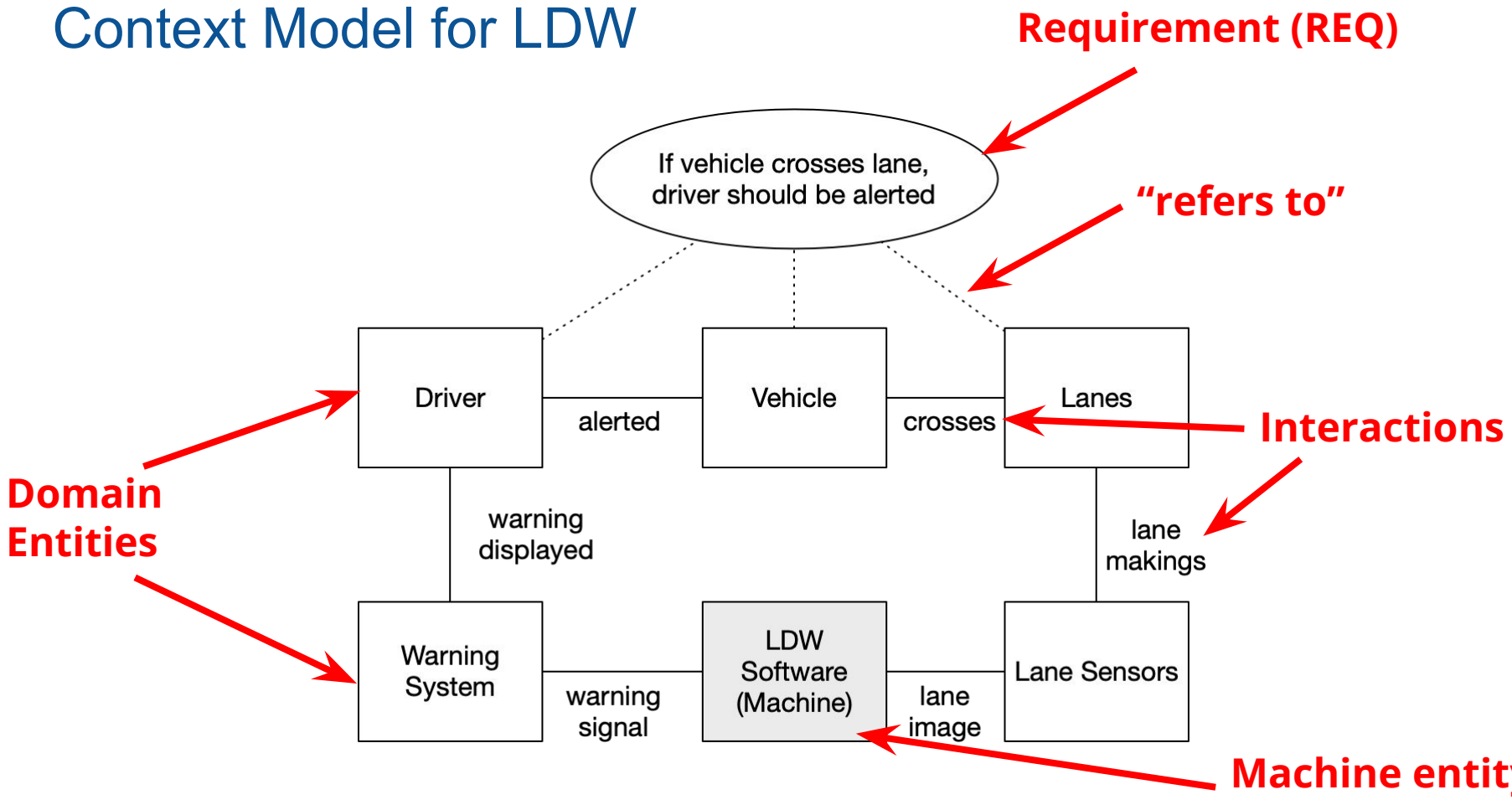
Satisfaction Argument



"If my software is implemented correctly (SPEC) and the world behaves as assumed (ASM), then the system achieves its requirement (REQ)"

Requirement (REQ): What the system must achieve, in terms of desired effects on the world
Specification (SPEC): What software must implement, expressed over the shared interface
Domain assumptions (ASM): What's assumed about the behavior/properties of the world; bridges the gap between REQ and SPEC

Context Model for LDW



A recipe for building a context model

1. State a requirement to be achieved by the system (REQ)
2. Identify entities that are referenced by the requirement
3. Identify other entities that interact with those entities in the real world
4. Connect domain entities to the software component
5. Design the specification (SPEC) on the software component that is needed to satisfy REQ
6. Identify domain assumptions (ASM) that are needed along with SPEC to satisfy REQ
7. Check whether any of the assumptions may be violated in practice
8. If so, relax ASM to reflect possible violations and design a new SPEC to ensure that SPEC /\ ASM => REQ

Case Study: Ambulance Dispatching System

Ambulance Dispatching System: Traditional Workflow

- Dispatcher receives an emergency 911 call and determines the location and severity of the incident
- Dispatcher looks up the list of nearby ambulances on a computer
- Dispatcher contacts and dispatches one of the available ambulances to the incident location
- Ambulance crew arrives at the location and treats the patient and/or transports them to a hospital

New, Automated Dispatching System

- **Automatic Dispatch Software**: The 911 operator enters the details of the incident into new software. The software decides which ambulance to allocate for the incident.
- **Automated Ambulance Localisation**: A GPS-based system is used to keep track of ambulances' locations.
- **Mobile Data Terminals**, installed inside each ambulance: The ambulance crew uses the terminal to communicate their status to the Automatic Dispatch Software (when they arrive at the incident scene, when they hand over the patient to a hospital, etc.,)
- **System requirement**: Ensure the arrival of an ambulance at an incident location within 15 min.

Breakout Activity

- **Task 1:** Develop a context model for the new ambulance dispatching system. Identify the list of domain assumptions (ASM) and software specifications (SPEC) that are needed to satisfy the requirement (REQ).
- **Task 2:** Share and describe your context model to another breakout group. Looking at the other team's context model, identify assumptions that may be violated in practice.
- **Task 3:** Based on the feedback from the other team, discuss how you would modify the specifications (SPEC) to deal with the violated assumptions (ASM).

Case Study: London Ambulance System (LAS)

Summary

- Domain assumptions are just as critical in achieving requirements
  - If you ignore/misunderstand these, your system may fail or do poorly (no matter how perfect your software is)
- Identify and document these assumptions as early as possible
- Some of the assumptions may be violated in practice
- The specification of the software should be designed with these assumptions & their possible violations in mind