Understand A Control System's Requirements and Its Specifications

7/25/2019

Presented by: Ruoxi Zhang

CS 846 – Summer, 2019

UNIVERSITY OF
WATERLOO

Outline

- Terminologies
  - Control system
  - Requirement and specification
  - Domain knowledge
- Example
- Conclusion

EXAMPLE OF A CONTROL SYSTEM (1)

A stainless steel turnstile is designed to provide unsupervised access control for office or building which realize intelligent pedestrian control and management.

https://www.indiamart.com/proddetail/card-reader-automatic-turnstile-16439201191.html

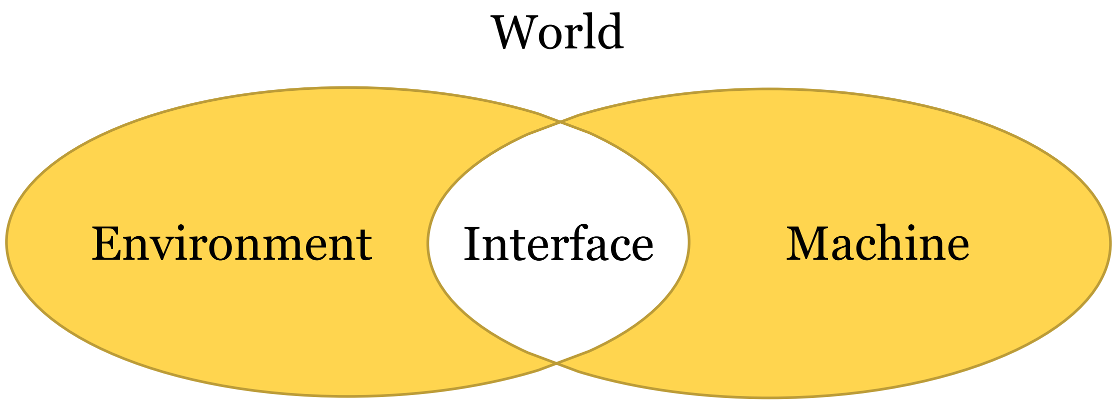
EXAMPLE OF A CONTROL SYSTEM (2)

A conventional fixed-wing aircraft flight control system consists of necessary operating mechanisms to control an aircraft's direction in flight.

https://www.123rf.com/photo_76903165_hong-kong-circa-november-2016-cockpit-of-emirates-airbus-a380-the-airbus-a380-is-a-double-deck-wide-.html

Control System

- A **control system** is a machine that interacts with its environment to bring about or maintain relationships in that environment [4].
  - **Machine**: the computer-based machine to be developed
  - **Environment**: the relevant physical world
  - **Interactions**: e.g., the machine receives input from the physical world via sensors and influences it via actuators



- Such systems are often deployed in safety-critical environments.

UNIVERSITY OF WATERLOO

Requirement vs. Specification (1)

- A **requirement** states desired relationships in the environment — relationships that will be brought about or maintained by the machine [4].
  - “What are we trying to accomplish?”
- A **specification** describes the behavior of the machine at its interface which sufficient to achieve the requirement [4].
  - “What would that software system do?”

Requirement vs. Specification (2)

- The environment is described in two ways [5]:
  - **Indicative** statements: describe the environment as it is in the absence of the machine
  - **Optative** statements: describe the environment as we hope it will become because of the machine
- R (a set of requirements) and S (a set of specifications) are optative.
- Both requirement and specification are expressed entirely in terms of environment phenomena.
  - Avoid implementation bias since no statements are made about the proposed machine [4].

Requirement vs. Specification (3)

- Each **action** must be identified as belonging to exactly one of the three categories.
  - All three types of actions are relevant to RE, and they might need to be described or constrained.

indicates which agent controls, performs, or takes responsibility for the action

Action Type	Environment Controlled	Machine Controlled
Shared	X	X
Unshared	X	~~X~~

[interface] observable by both the machine and environment

private to the environment and unobservable by the machine

internal actions of the machine

Specification

- A specification is derived from a requirement, and it is a restricted kind of requirement.
  - Remove all references to machine inaccessible phenomena.

Action Type	Environment Controlled	Machine Controlled
Shared	X	X
Unshared	X	

- Three rules of specifications [5]
  - The members of specifications do not constrain the environment;
  - They are not stated in terms of any unshared actions or state components;
  - They do not refer to the future.

Domain Knowledge (1)

- K: relevant domain knowledge (assumptions about the environment).
- K is indicative.
- Each property or assertion must be identified as a requirement (R), statement of domain knowledge (D), or specification (S).
- S and K together must be sufficient to guarantee that the requirements are satisfied: $S, K \vdash R$ [1, 5].

Domain Knowledge (2)

- Requirements that constrain the environment are satisfied by coordinating specifications with domain knowledge.

- Requirements with unshared information are refined using domain knowledge relating unshared information to shared information.

Action Type	Environment Controlled	Machine Controlled
Shared	X	X
Unshared	X	

UNIVERSITY OF WATERLOO

Domain Knowledge (3)

- The machine is under no obligation if it is used in an environment in which the assumption is false.

- E.g. specify the control program of a room heating system [2, 3]
  - Specification: corrective action should follow when the temperature sensor indicates that some limit value has been exceeded
  - Domain knowledge: the assumptions about the accuracy of the sensors

Example (1)

- E.g., specify the control system of a turnstile at the entrance to a zoo [4]
  - Mechanical hardware: a rotating barrier, a coin slot, and a locking device
  - Development job: controlling software
- Relevant environment phenomena

Unary predicates	**Designations**	**Share**	**Control**
Push(e)	In event e a visitor pushes the barrier to its intermediate position	Y	Env.
Enter(e)	In e a visitor pushes the barrier fully and so gains entry	N	Env.
Coin(e)	In e a valid coin is inserted into the coin slot	Y	Env.
Lock(e)	In e the turnstile receives a locking signal	Y	Machine
Unlock(e)	In e the turnstile receives an unlocking signal	Y	Machine

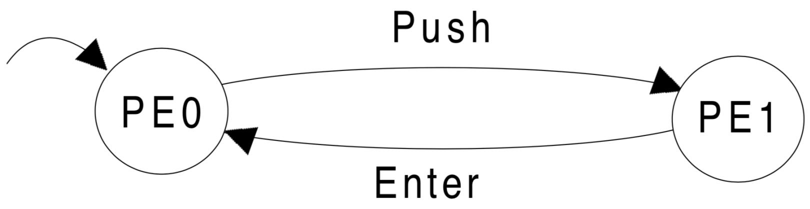
Example (2)

- Assume events are atomic and instantaneous.

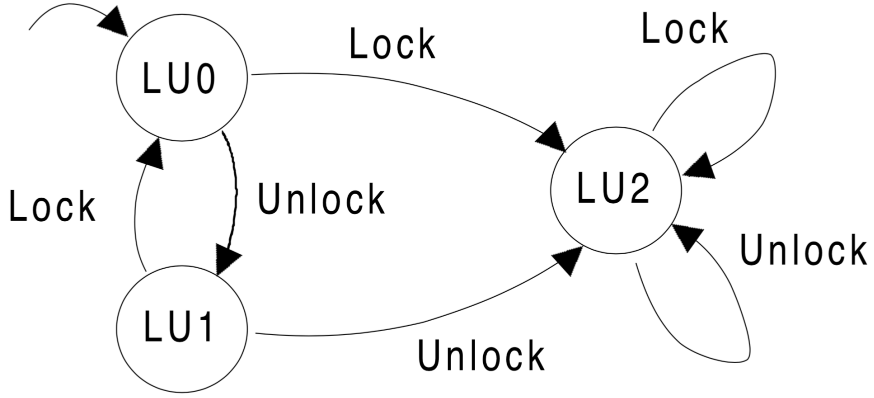
Event(e)	e is an atomic instantaneous event
Ends(e, v)	Event e ends interval v

Example (3)

- Domain Knowledge
  - [IND1] Push and Enter events alternate, starting with Push.



  - [IND2] $\forall e, v \cdot ((LU0(v) \wedge Ends(e, v)) \rightarrow \neg Push(e))$
    - If Lock and Unlock events alternate, starting with Unlock, then a Push event can occur only after an Unlock and before the next Lock.
    - Push events are impossible in state $LU0$.



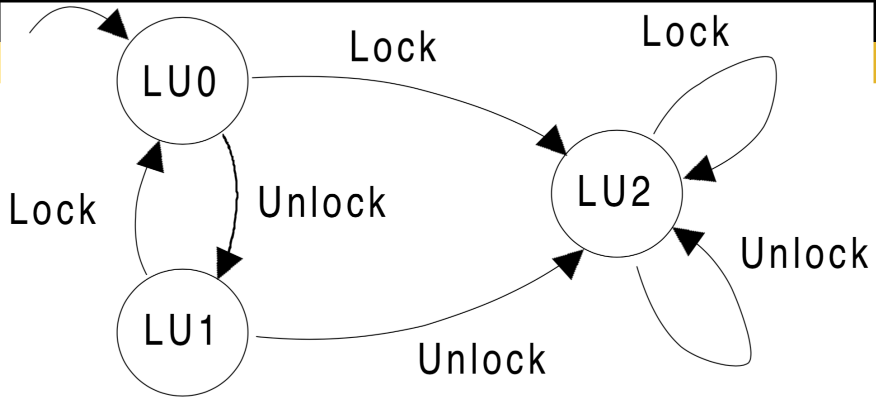
UNIVERSITY OF WATERLOO

Example (4)

- Requirement: no-one should enter without paying.
  - Note that “payments alternate with entries” is not a requirement.
  - [OPT1] $\forall\, v, m, n \cdot ((\mathrm{Enter\#}(\mathrm{v}, \mathrm{m}) \wedge \mathrm{Coin\#}(\mathrm{v}, \mathrm{n})) \rightarrow (m \leq n)))$ Enter# ≤ Coin#
    - Define Push#(v, n), Enter#(v, n) and Coin#(v, n) meaning that the count of Push, Enter and Coin events respectively preceding interval v is n.

- Problems
  - The enter events are not shared phenomena.
  - The specification constrains the state in every interval, including those that are in the future.

Example (5)



- Specification
  - [OPT1] $\forall\, v, m, n \cdot ((\text{Enter\#}(v, m) \wedge \text{Coin\#}(v, n)) \rightarrow (m \leq n)))$ — Enter# ≤ Coin#
  - [IND3] $\forall\, v, m, n \cdot ((\text{Enter\#}(v, m) \wedge \mathit{Push\#}(v, n)) \rightarrow (n - 1 \leq m \leq n)))$ — Push#-1 ≤ **Enter#** ≤ **Push#**
  - [OPT1a] $\forall\, v, m, n \cdot ((\text{Push\#}(v, m) \wedge \text{Coin\#}(v, n)) \rightarrow (m \leq n)))$ — **Push# ≤ Coin#**
  - When Push# already equals Coin#, the machine must prevent a further Push at least until after a further Coin event.
  - [OPT2-safety] $\forall\, v, e, n \cdot (((LU0(v) \wedge Push\#(v, n) \wedge Coin\#(v, n) \wedge Ends(e, v)) \rightarrow \neg Unlock(e))$
  - [DEF2-liveness] $ReqLock(v) \triangleq (LU1(v) \wedge \exists n \cdot (Push\#(v, n) \wedge Coin\#(v, n)))$
    - In state *ReqLock* the machine must perform a Lock event.

Conclusion

- The specification is implantable if [5]:
  - There is a set R of requirements (validated as acceptable to the customer).
  - There is a set K of statements of domain knowledge/assumptions (validated as true of the environment).
  - There is a set S of specifications.
    - Do not constrain the environment;
    - Do not consist of unshared actions;
    - Do not refer to the future.
  - A proof shows that $S, K \vdash R$.
  - A proof shows that S and K are consistent.

Together imply that S, K, and R are consistent with each other.

References

1. Hadar, I., Zamansky, A. & Berry, D.M. Empir Software Eng (2019). https://doi.org/10.1007/s10664-019-09718-5
2. Hayes, I., Jackson, M., Jones, C.: Determining the specification of a control system from that of its environment. In Araki, K., Gnesi, S., Mandrioli, D., eds.: Formal Methods (FME). Volume 2805 of LNCS. Springer, Berlin, DE (2003) 154–169
3. Jones, C.B., Hayes, I.J., Jackson, M.A.: Deriving specifications for systems that are connected to the physical world. In Jones, C.B., Liu, Z., J., W., eds.: Formal Methods and Hybrid Real-Time Systems. Volume 4700 of LNCS. Springer, Berlin, DE (2007)
4. Michael Jackson and Pamela Zave. 1995. Deriving specifications from requirements: an example. In Proceedings of the 17th international conference on Software engineering (ICSE '95). ACM, New York, NY, USA, 15-24. DOI=http://dx.doi.org/10.1145/225014.225016
5. Zave, P., Jackson, M.: Four dark corners of requirements engineering. ACM Transactions on Software Engineering and Methodology (TOSEM) 6 (1997) 1–30

THANK YOU