Design and Analysis of Distributed Interacting Systems

Lecture 11 – MSDs and Controller Synthesis

Prof. Dr. Joel Greenyer

June 27, 2013
Last Time: Satisfying an MSD Specification, Implementation, Realizability

• When does the controller $C$ of an object system **satisfy** an MSD specification $MS$?
  – What means $C \models MS$?
  – one controller that describes the behavior of the system: “closed world”

• When does a controller $C_{sys}$ of the system objects **implement** $MS$?
  – $\forall C_{env} : (C_{env} \parallel C_{sys}) \models MS$
  – system controller in every possible environment: “open world”

• Is $MS$ **realizable**:
  – does there exist an implementation?
  – $\forall C_{env} \exists C_{sys} : (C_{env} \parallel C_{sys}) \models MS$

*with restrictions imposed by the synchrony assumption*
Is the following MSD specification realizable?

Who says that, for example, not arrivedAtTable or blankArrived occurs when expecting arrivedAtPress here? (this would lead to a safety violation)
• Sometimes we have to assume that not arbitrary things can happen in the environment
  – environment assumptions or domain knowledge

• Idea: Formulate those with MSDs as well!

• Example: Assumptions in the Production Cell
  – if arm A is ordered to move to the press, it will eventually arrive at the press
    (unless it is ordered to move back to the table while it is “on its way” to the press)
Last Time: MSD Specification with Environment Assumptions

• We extend the definition of an MSD specification

\[ MS = (O, \Sigma, D) \]

- \[ A \cup G = D, A \cap G = \emptyset \]

• Let \( L(A) \) be the runs accepted by all assumption MSDs, and \( L(G) \) be the runs accepted by all requirement MSDs

• A controller \( C \) for the object system \( O \) satisfies an MSD specification iff

\[ L(C) \subseteq (\Sigma^\omega \setminus L(A)) \cup L(G) \]

– For all runs of the controller: the run either satisfies all MSDs in \( G \) or fails to satisfy at least one MSD in \( A \)

• “if a run does not satisfy the assumptions it needs not satisfy the requirements”
Last Time: MSD Specification with Environment Assumptions

- We extend the definition of an MSD specification:
  \[ MS = (O, \Sigma, D) \]
  \[ A \cup G = D, \quad A \cap G = \emptyset \]

- Let \( L(A) \) be the runs accepted by all assumption MSDs, and \( L(G) \) be the runs accepted by all requirement MSDs.

- A controller \( C \) for the object system \( O \) satisfies an MSD specification iff
  \[ L(C) \subseteq (\Sigma^\omega \setminus L(A)) \cup L(G) \]
  - For all runs of the controller: the run either satisfies all MSDs in \( G \) or fails to satisfy at least one MSD in \( A \)
    - “if a run does not satisfy the assumptions it needs not satisfy the requirements”

- Usually, assumptions express what can happen in the environment (spontaneously) or how the environment must in turn react to system events.

- Usually, requirements express how the system must react to environment events.

- If a run satisfies the assumptions, it must also satisfy the requirements:
  \[ \pi \models MS \iff \pi \models A \implies \pi \models G \]
Last Time: The Play-Out Algorithm

- Our goal is to find a controller for $O_S$ so that the global controller formed with any possible controller for objects $O_E$ satisfies the MSD specification (under the synchrony assum.)

- Idea: Execute the MSDs!
  - Wait for any environment event to occur
  - While there are executed system messages enabled, execute one if this does not lead to a safety violation
  - Repeat the process.

what does play-out here?

not formulated precisely before
Last Time: ScenarioTools Play-Out

(Demo)
Last Time: ScenarioTools Play-Out

leads to cold violation in assump. MSD

progresses c/e and h/e cut in active assump. MSD

initializes new req. MSD

but would lead to safety violation

progresses h/m cut in req. MSD
• Often systems consist of many components with changing relationships, e.g. RailCab:
  – which RailCab is in front?
  – which track section is a RailCab on?
  – is the RailCab currently in a station?

• The relationships influence what may/must happen
  – the relationships determine which object is responsible to do what (which “role” does which object play?)
  – certain things may/must happen only in certain situations (behavior can be “context sensitive”)
Dynamic Systems and Dynamic Bindings

- So far, a lifeline represented exactly one object.
- Often we would like that the same scenario applies for different combinations of objects.
  - Lifelines should bind dynamically to objects.

- Example: static binding.
Dynamic Systems and Dynamic Bindings

- So far, a lifeline represented exactly one object
- Often we would like that the same scenario applies for different combinations of objects
  - lifelines should bind dynamically to objects

- Example:
  - dynamic binding

```
(dynamic) object system

next := rc.current.next
```

Objects with links:
- `rc: RailCab`
- `next: TrackSectionControl`

Some links can change:
- `registered-RailCabs`
- `current`
- `next`
Dynamic Systems and Dynamic Bindings

- So far, a lifeline represented exactly one object
- Often we would like that the same scenario applies for different combinations of objects
  - lifelines should bind dynamically to objects

Example: dynamic binding

- lifelines typed by classes
- some links can change
- objects with links
- sending and receiving lifelines of first message bind upon message unification; binding expressions determine which objects the other lifelines shall bind to (here: OCL)
Dynamic Systems and Dynamic Bindings

- The dynamic binding can also exploit generalization relationships

In the class diagram:

```
TrackSectionControl

CrossingControl
```

```
 MSD RequestEnterAtEndOfTrackSection

env:Environment | rc:RailCab | next:TrackSectionControl

endOfTS requestEnter enterAllowed

lastBrake (isAllowed)

next := rc.current.next
```

i.e., this requirement holds also if the next track section is a crossing.
Dynamic Systems and Dynamic Bindings

- **enterAllowed**: Different behavior in different situations

In the class diagram:

1. **MSD RequestEnterAtEndOfTrackSection**
   - `env:Environment`
   - `rc:RailCab`
   - `next:TrackSectionControl`

   - `endOfTS` to `requestEnter`
   - `lastBrake` to `enterAllowed (isAllowed)`

   - `next := rc.current.next`

2. **MSD CloseBarriers**
   - `rc:RailCab`
   - `crc:CrossingControl`
   - `b:Barriers`

   - `requestEnter` to `closeBarriers`

   - `alt`
     - `enterAllowed (true)` to `barriersClosed`
     - `enterAllowed (false)` to `barriersBlocked`

   - `b := crc.barriers`
**Parametrized Messages**

- **enterAllowed**: Different behavior in different situations

The parameter is not specified, `isAllowed` is an *unbound* variable. The scenario shall consider each possible value.

If the parameter is not specified, the message is also called *symbolic*, otherwise the message is *concrete*.

Here a concrete value is specified.
Previously: Message Unification

- A message event in the system can be *unified* with a message in an MSD (called diagram message) if
  - the message names are equal
  - the source object is represented by the source lifeline
  - the target object is represented by the target lifeline
Message Unification and Parametrized Messages

- A message event in the system can be *parameter unified* (or is *parameter unifiable*) with a message in an MSD iff
  - they are unifiable *and*
  - the diagram message is *symbolic* *or*
  - the diagram message is *concrete* and the parameter value carried by the message event is equal to the parameter value specified for the diagram message

- Violation of parameterized messages:
  It is a *violation* if a message event occurs that cannot be parameter unified with an enabled parameterized message, but can be unified with a message in the MSD
  - a safety/cold violation depending on cut temperature
Message Unification and Parametrized Messages

• Violation of parameterized messages:
  It is a violation if a message event occurs that cannot be parameter unified with an enabled parameterized message, but can be unified with a message in the MSD
  – a safety/cold violation depending on cut temperature

The power of modalities:
  another MSD can still require that the RailCab is not allowed to enter the crossing (maybe due to another problem)
Message Unification and Parametrized Messages

- Violation of parameterized messages:
  It is a *violation* if a message event occurs that cannot be parameter unified with an enabled parameterized message, but can be unified with a message in the MSD
  - a safety/cold violation depending on cut temperature

The power of modalities: another MSD cannot require that the RailCab is allowed to enter the crossing!

Safety violation if enterAllowed(true) occurs (or requestEnter, closeBarriers, barriersClosed, barriersBlocked)
Message Unification and Parametrized Messages

- Violation of parameterized messages:
  It is a violation if a message event occurs that cannot be parameter unified with an enabled parameterized message, but can be unified with a message in the MSD
  - a safety/cold violation depending on cut temperature

```
MSD RequestEnterAtEndOfTrackSection

env:Environment  rc:RailCab  next:TrackSectionControl

endOfTS  requestEnter  enterAllowed(isAllowed)

safety violation if endOfTS, requestEnter, or lastBreak occurs (enterAllowed(true) and enterAllowed(false) is fine)
```
Design and Analysis of Distributed Interacting Systems

Lecture 11 – MSDs and Controller Synthesis

Prof. Dr. Joel Greenyer

June 27, 2013
Agenda

- Environment & System Controllers (✓)
- Realizability (✓)
- Environment Assumptions (✓)
- The Play-Out Algorithm (✓)
- Advanced concepts:
  - Dynamic Object Systems and Dynamic Lifeline Binding (✓)
  - Parametized Messages (✓)
- Controller Synthesis
Realizability – Example

• what now? realizable?

I think so...

this can be very hard to “see”, i.e. to check manually
Realizability – Example

• what now? realizable?

I think so...

this can be very hard to “see”, i.e. to check manually

we have to find an implementation or show that there is none...

... can we do this automatically?

...
A Game-Theoretic Perspective

- Reformulate “Does there exist an implementation?” as
- “Can the system win the following game?”:
  - The environment can send arbitrary sequences of events
  - The system reacts to each environment event by sending finite sequences of system events (synchrony assumption)
  - There must be no safety or liveness violation in any requirement MSDs
  - Or there is a safety or liveness violation in an assumption MSD
The Game Graph

- LTS of reachable configuration of cuts of active MSDs

- System events: controllable
- Env. events: uncontrollable

- Safety violation in assumptions
- Safety violation in requirements

- NoBlankArrivesBeforeArmAReturnedToTable (2, 2, 1)
  ArmATransportBlankToPress (-1, 2, 1)

- NoBlankArrivesBeforeArmAReturnedToTable (2, 3, 2)
  ArmATransportBlankToPress (-1, 4, 3)
  ArmAMoveFromTableToPressAssumption (2, 2)

- ArmATransportBlankToPress (-1, 2, 1)

- ArmATransportBlankToPress (-1, 3, 2)

- ArmATransportBlankToPress (-1, 5, 4)
The Game Graph

- LTS of reachable configuration of cuts of active MSDs

![Diagram]

- System events: controllable
- Env. events: uncontrollable

- Safety violation in assumptions
- Safety violation in requirements
The Game Graph

- LTS of reachable configuration of cuts of active MSDs

![Diagram of system events and environment events](image)

- System events: controllable
- Environment events: uncontrollable

- Safety violations in assumptions
- Safety violations in requirements

- NoBlankArrivesBeforeArmAReturnedToTable (2, 2, 1)
- ArmATransportBlankToPress (-1, 3, 2)
- ArmAMoveFromTableToPressAssumption (2, 2)
- NoBlankArrivesBeforeArmAReturnedToTable (2, 3, 2)
- ArmATransportBlankToPress (-1, 4, 3)

- BLANK ARRIVED, ARRIVED AT TABLE
- ARRIVED AT PRESS

- System events: controllable
- Environment events: uncontrollable

- Safety violations in assumptions
- Safety violations in requirements

- NoBlankArrivesBeforeArmAReturnedToTable (2, 2, 1)
- ArmATransportBlankToPress (-1, 5, 4)
The Game Graph

- LTS of reachable configuration of cuts of active MSDs

The diagram illustrates the game graph for the system, showing transitions between states such as:

- blankArrived → pickUp → moveToPress → arrivedAtPress
- arrivedAtTable → moveToTable → arrivedAtPress
- ArmATransportBlankToPress

The graph also highlights assumptions and environment events, such as:

- NoBlankArrivesBeforeArmAReturnedToTable (2, 2, 1)
- ArmAMoveFromTableToPressAssumption (2, 2)

There are also safety violations indicated, such as:

- safety violation in assumptions
- safety violation in requirements

The system events are controllable, while the environment events are uncontrollable.
The Game Graph

- LTS of reachable configuration of cuts of active MSDs

...
The Game Graph

- LTS of reachable configuration of cuts of active MSDs

System events: controllable
Env. events: uncontrollable

Safety violation in assumptions
Safety violation in requirements

...
The Game Graph

- LTS of reachable configuration of cuts of active MSDs

 system events: controllable
 env. events: uncontrollable

blankArrived, arrivedAtTable
safety violation in assumptions
safety violation in requirements
The Game Graph

- LTS of reachable configuration of cuts of active MSDs

- NoBlankArrivesBeforeArmAReturnedToTable (2, 2, 1)
- ArmATransportBlankToPress (-1, 2, 1)
- blankArrived...
- NoBlankArrivesBeforeArmAReturnedToTable (2, 3, 2)
- ArmATransportBlankToPress (-1, 4, 3)
- ArmAMoveFromTableToPressAssumption (2, 2)

- arrivaAtPress
- blankArrived, arrivedAtTable

- safety violation in assumptions
- safety violation in requirements

- system events: controllable
- env. events: uncontrollable

- <<EnvironmentAssumption>>
- ArmAMoveFromTableToPressAssumption

- controller
- controller
- controller

- (active copy terminated)
The Winning Condition

- **Goal states:**
  - There must be no safety or liveness violation in any requirement MSDs.
  - Or there is a safety or liveness violation in an assumption MSD.

- The system can win the game if it can guarantee that goal states are visited infinitely often, regardless of the moves of the environment:
  - the system can choose controllable transitions (i.e., sending some system message)
  - but the system must assume that the environment can choose any uncontrollable transition (i.e., send any environment messages)

```
NoBlankArrivesBeforeArmAReturnedToTable (2, 2, 1)
ArmATransportBlankToPress (2, 1, -1)
```

Not covered in that lecture, see slides of the next lecture.
The Winning Condition

- **Goal states:**
  - There must be no safety or liveness violation in any requirement MSDs
  - Or there is a safety or liveness violation in an assumption MSD

- The system can win the game if it can guarantee that goal states are visited infinitely often, regardless of the moves of the environment
  - the system can choose controllable transitions (i.e., sending some system message)
  - but the system must assume that the environment can choose any uncontrollable transition (i.e., send any environment messages)

Games with a winning condition requiring that some condition must hold infinitely often are also called Büchi Games.

Not covered in that lecture, see slides of the next lecture
Winning Strategy

- A **winning strategy** is a function that maps states to sets of controllable events that the system can choose in these states in order to win.
  - set of controllable events is also called *winning moves*.
  - empty set: the system cannot choose (it's the environment's turn).
  - more than one winning move: the strategy is non-deterministic, i.e., there are multiple ways to win.

---

Not covered in that lecture, see slides of the next lecture.
Check if Winning Strategy Exists: Algorithm Sketch

- Given: Game graph with initial state $q_0$ and goal states $Goal$
  - we assume that $q_0 \in Goal$

- Two functions:
  - boolean isGoalReachable(State $q$)
    - checks if the system can guarantee to reach goal states from $q$
  - boolean solveBüchiGame(State $q$) – called with $q_0$

Set<State> $G = Goal$
Set<State> $Win = \emptyset$

foreach $g \in G$: if (isGoalReachable($g$)) then $Win.add(g)$
while $G \neq Win$ do
  $G = Win$
  $Win = \emptyset$
  foreach $g \in G$: if (isGoalReachable($g$)) then $Win.add(g)$
endwhile
return $q_0 \in G$

Not covered in that lecture, see slides of the next lecture

successively eliminates goal state from which the system cannot reach other goal states

Check if Winning Stratgy Exists: Algorithm Sketch

- Given: Game graph with initial state $q_0$ and goal states $Goal$
  - we assume that $q_0 \in Goal$
- Two functions:
  - boolean $isGoalReachable(State \ q)$
    - checks if the system can guarantee to reach goal states from $q$
  - boolean $solveBuchiGame(State \ q)$ called with $q_0$

```java
Set<State> G = Goal;
Set<State> Win = ∅;
foreach g ∈ G: if (isGoalReachable(g)) then Win.add(g)
while G ≠ Win do
    G = Win
    Win = ∅
    foreach g ∈ G: if (isGoalReachable(g)) then Win.add(g)
endwhile
return $q_0 \in G$
```


not covered here: when exploring the game graph, the procedure checks for all explored states whether the system can reach a goal state from it. From this information we in the end extract the winning strategy.

Not covered in that lecture, see slides of the next lecture
ScenarioTools Controller Synthesis

- Game graph for the production cell example (only Arm A)

Not covered in that lecture, see slides of the next lecture.
Realizability – Example

- what now? realizable?

yes!

Not covered in that lecture, see slides of the next lecture
ScenarioTools Controller Synthesis

- The controller extracted from the game graph

Not covered in that lecture, see slides of the next lecture
ScenarioTools Controller Synthesis

- Example from the last assignment

Not covered in that lecture, see slides of the next lecture

not realizable if we assume a global acceptance condition
ScenarioTools Controller Synthesis

- Example from the last assignment

```
• Example from the last assignment

Not covered in that lecture, see slides of the next lecture
```

```
initial state
```

```
green: winning state
from here the system can guarantee to reach a goal state
```

```
it turns out that the system cannot guarantee to reach a goal state infinitely often: no winning strategy and no controller exists
```
ScenarioTools Controller Synthesis

• Another technical example

Not covered in that lecture, see slides of the next lecture

realizable
ScenarioTools Controller Synthesis

- The game graph

Not covered in that lecture, see slides of the next lecture
The controller

Not covered in that lecture, see slides of the next lecture

layout is a bit different, unfortunately

initial state
Agenda

• Environment & System Controllers (✓)
• Realizability (✓)
• Environment Assumptions (✓)
• The Play-Out Algorithm (✓)
• Advanced concepts:
  – Dynamic Object Systems and Dynamic Lifeline Binding (✓)
  – Parametized Messages (✓)
• Controller Synthesis (✓)

Not covered in that lecture, see slides of the next lecture