ROBOSTAR

Matt Windsor
Ana Cavalcanti

RoboCert: Property Specification for Robotics using Sequence Diagrams

Why is property specification important?

Robot failures are costly and dangerous! We must ensure that robot software and hardware behaves according to the properties we need:

Universal property
'if the robot finds an obstacle, it will *always* turn around'

Existential property
'it is *possible* that the robot finds an obstacle then turns'

Negated property
'if the robot finds an obstacle, it *will not* go straight on'

Robot properties are **often time-sensitive**:

- deadlines ('the robot will *immediately* turn around')
- budgets ('the robot *can wait 1s* before turning')

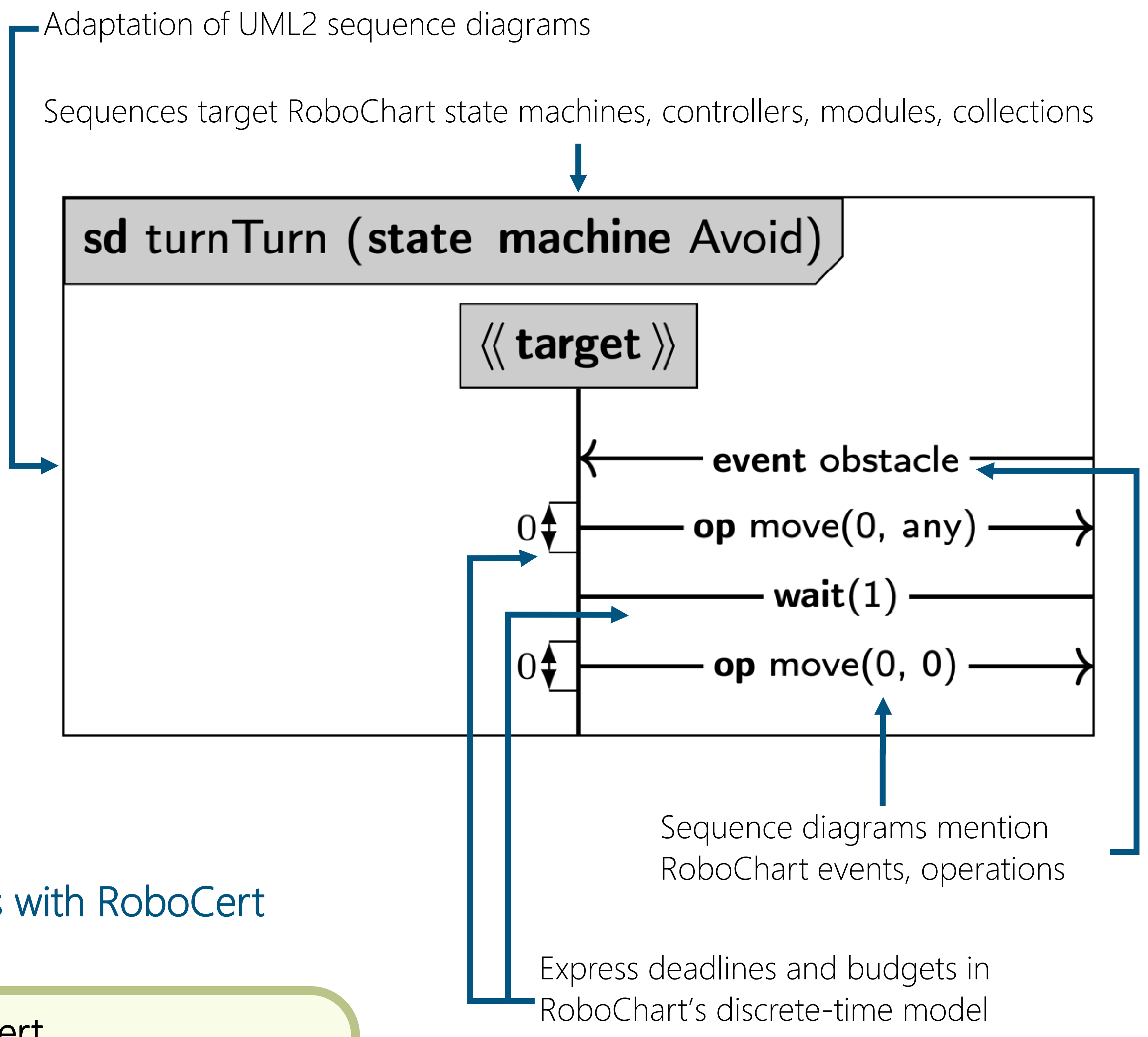
Can't we just use English?

English is ambiguous! In the property 'when the robot finds an obstacle, it waits 4s then turns around', can the robot do other things while it waits? Can it turn around in that time?

What about mathematics?

Formal mathematical languages are good for proving properties, but can be **hard to understand** for roboticists.

What do our sequence diagrams look like?



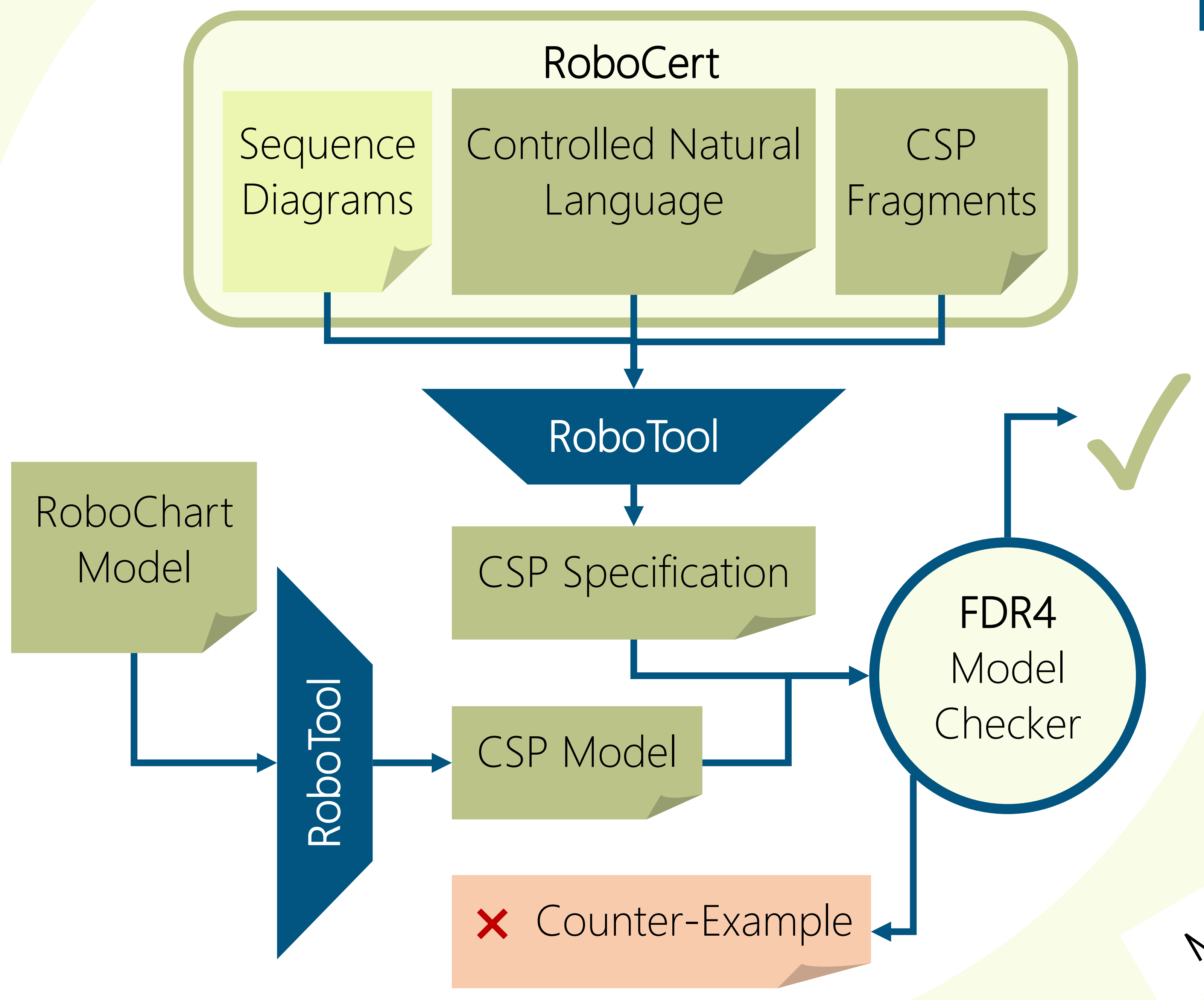
Adaptation of UML2 sequence diagrams

Sequences target RoboChart state machines, controllers, modules, collections

Sequence diagrams mention RoboChart events, operations

Express deadlines and budgets in RoboChart's discrete-time model

Verifying Properties with RoboCert



Our approach

We use **sequence diagrams** (with a formal mathematical meaning) to capture properties in an *unambiguous but understandable* way.

Future directions

More case studies

Hybrid properties over both software and hardware, dealing with both discrete and continuous time

System and scenario-level properties we can lower to properties over parts of RoboChart/RoboSim models

Graphical tools for viewing and creating sequence diagrams

Features

- supports the RoboChart **component model**, with properties over single components or collections of components
- messages over RoboChart **events and operations**
- **deadlines and budgets** using RoboChart's discrete time model
- **UML control flow**: loops, alternative choice, and optional choice
- 'do anything in a set of given messages **until**...' control-flow
- **liveness reasoning** through 'hot/cold' (must occur/may occur) modality on messages, and an 'xalt' mandatory alternative choice
- automated generation of CSP processes for **verification** using the FDR4 model checker

robostar.cs.york.ac.uk

University of York · Royal Academy of Engineering · UKRI Verifiability Node · UKRI Engineering and Physical Sciences Research Council