

Compositional Correctness by Dezyne

MDENET Symposium

Rutger van Beusekom (CTO Verum)¹

December 1st, 2022



¹rutger{@dezyne.org,.van.beusekom@verum.com}

Simon (Sinek) Says: "Start with Why"

- Why: Make Software Design an Engineering Discipline
- How: Take Formal Methods Mainstream
- What: Compositional Correctness by Dezyne

Programs as Statements

- Transform Input to Output
- Finish
- Produce a Single Result
- Sequential
- Deterministic
- Sum of its parts

Programs as Statements

- Transform Input to Output
- Finish
- Produce a Single Result
- Sequential
- Deterministic
- Sum of its parts

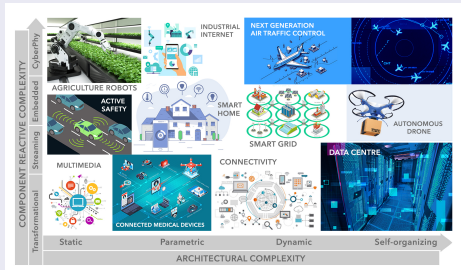
Software Systems

- Transform(State) Input to Output
- Continuous
- Reactive
- Concurrent
- Non-deterministic
- Emergent behaviour

Our (Collective) Challenge in Engineering the Systems of the Future

Resolving Conflicting Demands

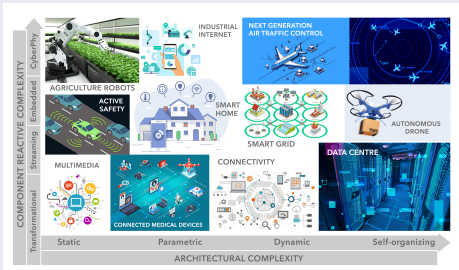
- Productivity
- Correctness



Our (Collective) Challenge in Engineering the Systems of the Future

Resolving Conflicting Demands

- Productivity
- Correctness



Manage Complexity through Rigorous Design

- Formalizing and Validating Requirements
- Design Top down
 - Stepwise Refinement
 - Verifiable
- Construct Bottom up
 - Composition
 - Correct by Construction

Language for Structure & Behaviour

The image shows a Visual Studio Code window titled "System View - Visual Studio Code". The left pane displays the source code for a state machine, and the right pane displays a corresponding state machine diagram.

Code (Left Pane):

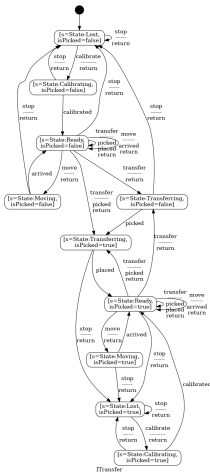
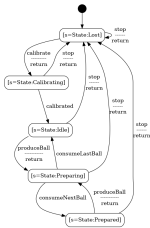
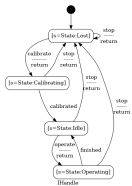
```
File Edit Selection View Go Run Terminal Help
MaterialHandler.dsn x MaterialHandler
315 @ComponentHandler
316
317 @requires Handle ctrl
318 @requires IfTransfer ready
319 @requires Inspect inspector
320 @requires IfFeed FeedPort
321 @requires ICoverly acceptPort
322 @requires ICoverly rejectPort
323
324
325
326
327 @state
328 state State {List: Calibrating, Idle, Operating}
329 subset useCount {0..4}
330 enum Operation {PreparingInput, TransferringInput, PreparingOutput, Transfer}
331
332 state s = state.list
333 useCount calibrated = 0
334 useCount requiresCalibrations = 3
335 Operation o = Operation.PreparingInput
336
337 bool isCalibrating = false
338 bool isBallAtInput = false
339 bool isRobotAtInput = false
340 bool isRobotAtOutput = false
341 bool isInspectorAtInput = false
342 bool isInspectorAtOutput = false
343 bool isBallAccepted = false
344 bool isAcceptEmpty = true
345 bool isRejectEmpty = true
346
347 @event prepareInputTransfer
348 o = Operation.PreparingInput
349 if (!isBallAtInput) {FeedPort.produceBall(); isBallAtInput = false}
350 if (!isInspectorAtInput) {inspector}
351 if (!isRobotAtInput) {sendRobotToInputPick(); isRobotAtInput = false}
352
353 @tryInputTransfer
354
355 @event sendRobotToInputPick
356 robot.move(|config.get("Position::Robot::X::InputPick")|)
357 |config.get("Position::Robot::Y::InputPick")|)
358 isRobotAtInput = false
359
360 @event sendRobotToOutputPick
361 o = Operation.PreparingOutput
362 isRobotAtOutput = false
363 robot.move(|config.get("Position::Robot::X::OutputPick")|)
364 |config.get("Position::Robot::Y::OutputPick")|)
365
366 @event finalizedPrepare
367 sendRobotToInputPick
368 if (!isBallAtInput) {isInspectorAtInput}
369 o = Operation.Finalizing
370 else
371 o = Operation.PreparingInput
372
373 @tryInputTransfer
374 if (!isBallAtInput && !isRobotAtInput && !isInspectorAtInput)
375 robot.transfer(|config.get("Position::Robot::X::InputDrop")|)
376 |config.get("Position::Robot::Y::InputDrop")|)
377 o = Operation.TransferringInput
378
379
380 @event tryOutputTransfer
381
```

Diagram (Right Pane):

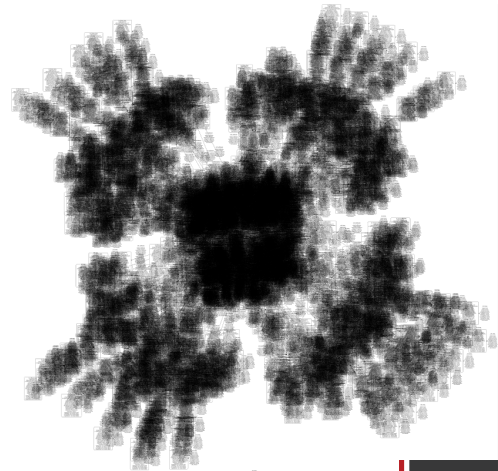
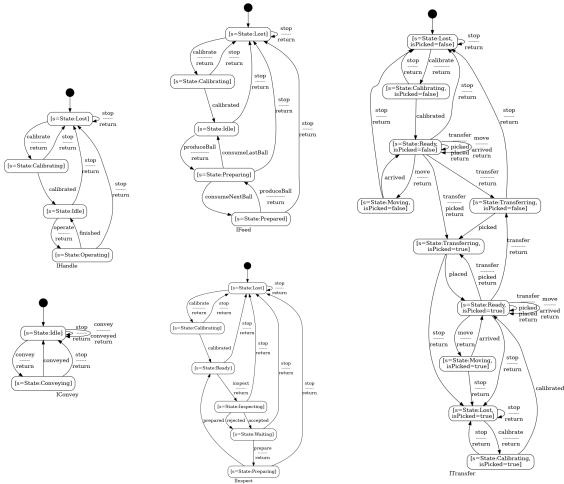
The diagram is a state machine diagram for the "Handler" component. It shows a hierarchy of states and transitions. The root state is "State". Transitions are triggered by events such as "prepareInputTransfer", "tryInputTransfer", "sendRobotToInputPick", "sendRobotToOutputPick", "finalizedPrepare", "tryInputTransfer", and "tryOutputTransfer". The diagram illustrates the complex logic of the state machine, including state changes and event handling.



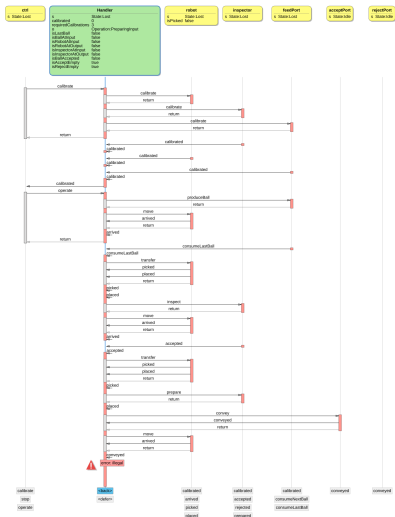
State Space



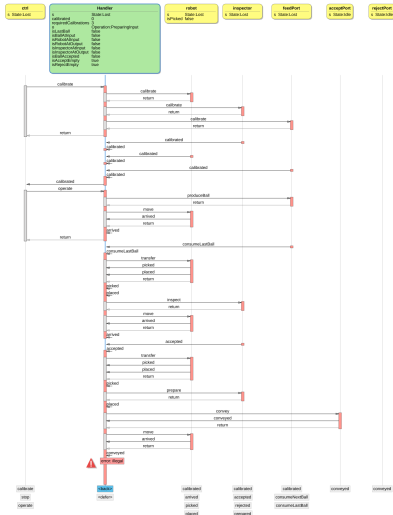
State Space Explosion



Model Checking and Divide and Conquer to the Rescue



Model Checking and Divide and Conquer to the Rescue



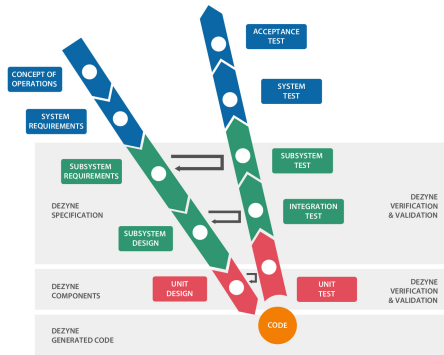
Compositional Correctness

- Interfaces: Design by Contract
- Components fully encapsulated by Interfaces
- Construct Systems from Verified Components



Multinationals

- Semiconductor
- Healthcare
- Automotive



Want to know more or give it a try?

Dezyne is FREE (as in freedom, not as in lunch) software

Visit

<https://dezyne.org>



<https://download.verum.com>



Want to know more or give it a try?

Dezyne is FREE (as in freedom, not as in lunch) software

Visit

<https://dezyne.org>



<https://download.verum.com>

Roadmap

- Constraining Interfaces (1st Release Candidate)
- Shared Interface State (Working prototype)
- Module Specification (Proof of concept)
- Functional Verification (Proof of concept)
- Data Contracts
- Legacy Transformation
- Model Based Refactoring



Q&A



Some Epigrams by Alan J. Perlis

- Every program is a part of some other program and rarely fits.
- It is easier to write an incorrect program than understand a correct one.
- A programming language is low level when its programs require attention to the irrelevant.
- Programmers are not to be measured by their ingenuity and their logic but by the completeness of their case analysis.
- Fools ignore complexity. Pragmatists suffer it. Some avoid it. Geniuses remove it.
- <http://www.cs.yale.edu/homes/perlis-alan/quotes.html>