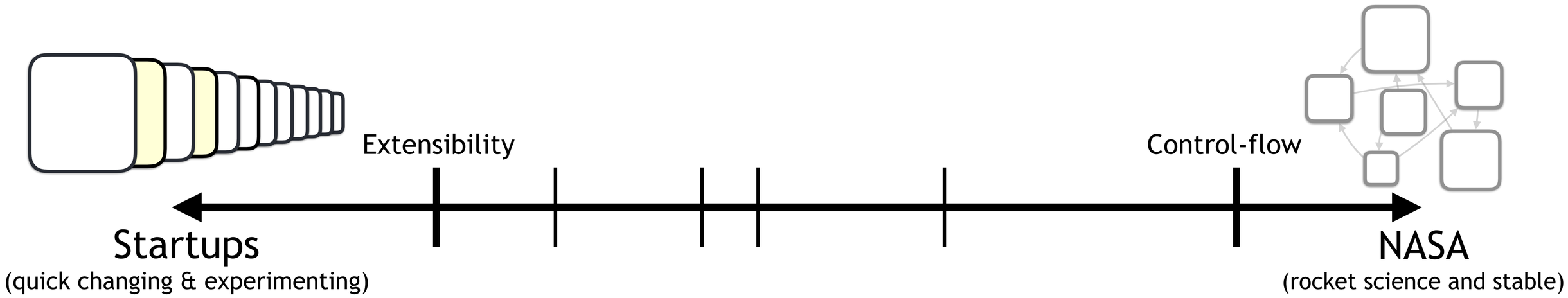
However I think that this approach comes with some cost. For instance `if (isMultipleOfThree(x) && endsWithDigitFive(x))` is very easy to understand, but `isMultipleOfThree(x)` and `endsWithDigitFive(x)` in different modules is a lot harder. Personally I see this more as a trade-off: **trading control flow for extensibility**.

Trade off spectrum



🌟 Future 🌟

System customization and personalization will become a new major phase in the life cycle of systems. End-users will be able to add specification units to control desired system behavior. These may be coded by the end users themselves or by the programmers supporting them, and perhaps even downloaded from large pools of vendor-provided and community-developed behaviors for similar kinds of systems.

🚧 Constraint solvers

Luca Matteis @lmatteis · Jan 9

An important feature of SBP is its intuitiveness and succinctness. These properties are a consequence of the ability to specify forbidden behavior directly and explicitly, rather than doing so using control-flow conditions, designed to prevent certain pieces of code or specification from actually doing the undesired action. In SBP, this feature was originally embodied in the use of concrete lists of requested events and filter-based blocking. Using this paper's extensions, this is done with constraint solvers. For example, one can now build and test the specification that a vehicle is not allowed to enter a road intersection when the traffic light is red, even before having coded how vehicles behave. Other

Show this thread

Luca Matteis @lmatteis · Jan 9

🎄 Xmas came late!

```
def forward_backward():
  while True:
    if dist > CLOSE:
      if dist < FAR:
        yield {Request([pL,pR]): pL = pR = MAX·(dist−CLOSE) / FAR−CLOSE}
      else:
        yield {Request([pL,pR]): pL = pR = MAX}

    else:
      if dist > VERY_CLOSE:
        yield {Request([pL,pR]): pL = pR = MAX·(dist−CLOSE) / CLOSE−VERY_CLOSE}
      else:
        yield {Request([pL,pR]):  pL = pR = −MAX}

def spin():
  while True:
    if abs(dir.error) > 3:
      if dir.error > 0:
```

1 ♡ 1

🎨 Live Sequence Charts

Luca Matteis
@lmatteis

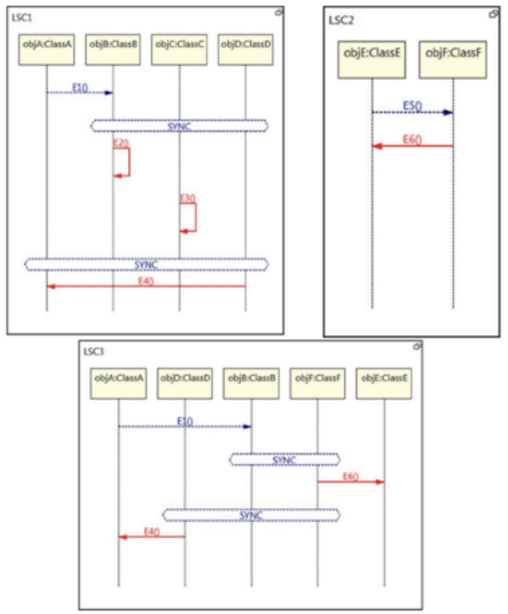


Fig. 4. Live Sequence Charts (LSCs) example: Scenario LSC1 specifies that after event E1 occurs, events E2 and E3 must occur, in any order, and, after both of them occur (enforced by the SYNC construct), E4 must occur. LSC2 specifies that after E5 occurs E6 must occur, and LSC3 specifies that once E1 occurs, E4 cannot occur until E6 occurs. Hence, when executing these LSCs, after E1 is triggered E4 will be delayed until E5 is triggered (by the environment or by some other scenario), subsequently triggering E6 and enabling E4.

🧤 Embedding with Statecharts

Luca Matteis
@lmatteis

Replying to @DavidKPiano @FacebookOrigami and @NoCodeConf

Indeed. Harel recently published an article regarding combining both approaches. They call for treating each parallel state as a scenario

One thing about Origami is that it was built without coding in mind. Hence there must be something particularly natural about that approach