National Aeronautics and Space Administration

A Modeling Language for Next Generation Flight Software

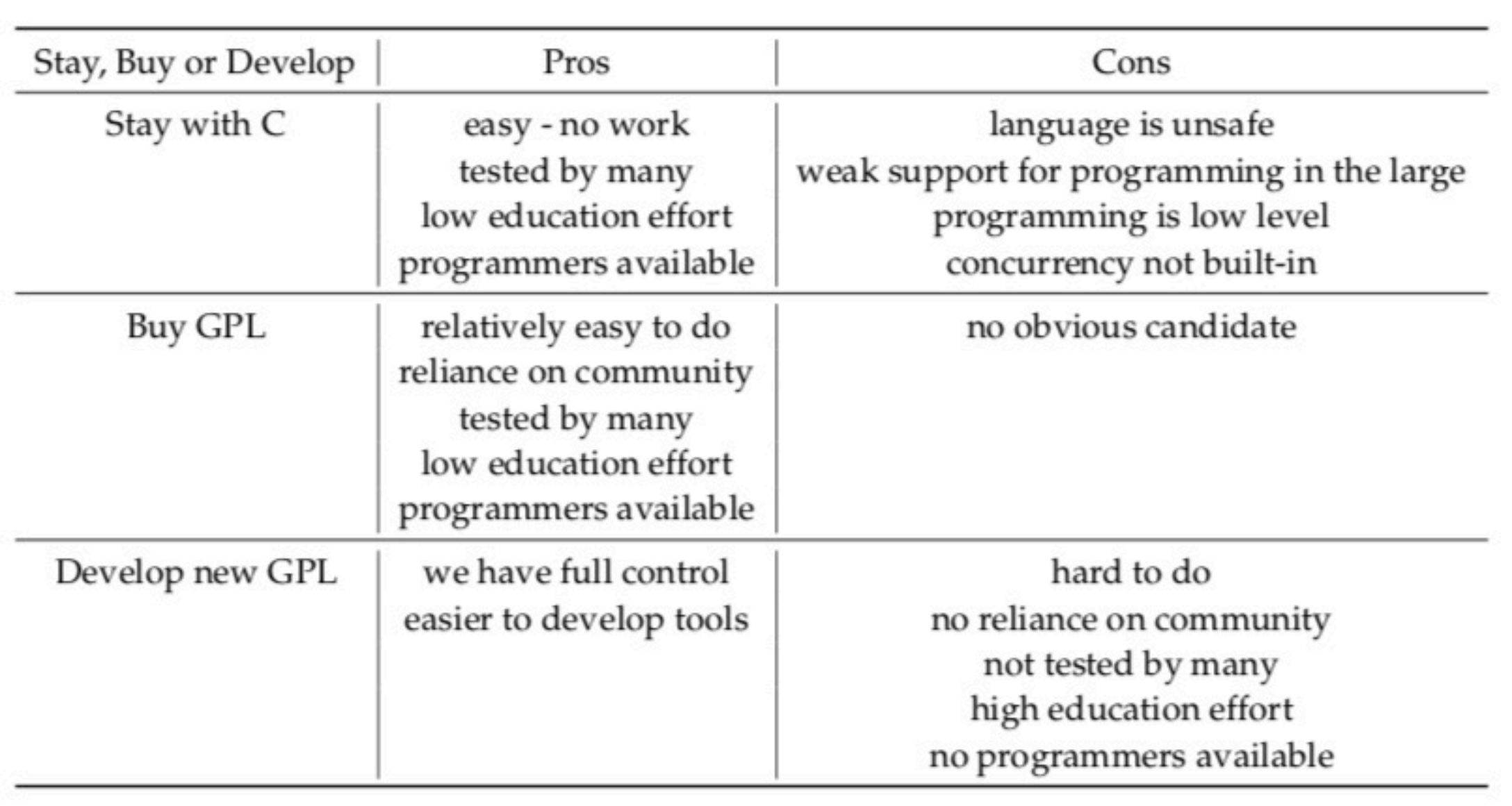
Principal Investigator: Klaus Havelund (348B)
Robert Bocchino (348C)
Program: Topic

Project Objective:

The objective of this work is to explore how to improve the reliability of flight software produced at JPL, as well as improving the productivity of programmers. This objective is to be seen in the context of JPL's increased focus on autonomy, which will result in development of increasingly complex software systems. Flight software is currently developed in the C programing language. We argue that this language is too low-level for the development of the next-generation complex autonomous software systems with millions of lines of code, and additionally is unsafe due to weak guarantees provided by the standard C compilers. Our goal is to replace C with a higher-level safer language for programming flight software. In addition, this language should be supported by advanced testing, monitoring, formal verification, and visualization technology to assist programmers in ensuring that programs are correct with respect to requirements. We also address the modeling activity which normally precedes coding. In the current approach to software design, early high level designs are described informally in Word and PowerPoint, and crucial implementation challenges are only uncovered when the developer starts to implement the informal design as low-level C code. This approach makes complex software hard and expensive to write, as crucial design flaws are often not uncovered until late testing of the C code, requiring late changes and workarounds. It is therefore an additional goal to provide a language in which flight software can be modeled as well as programmed.

Benefits to NASA and JPL:

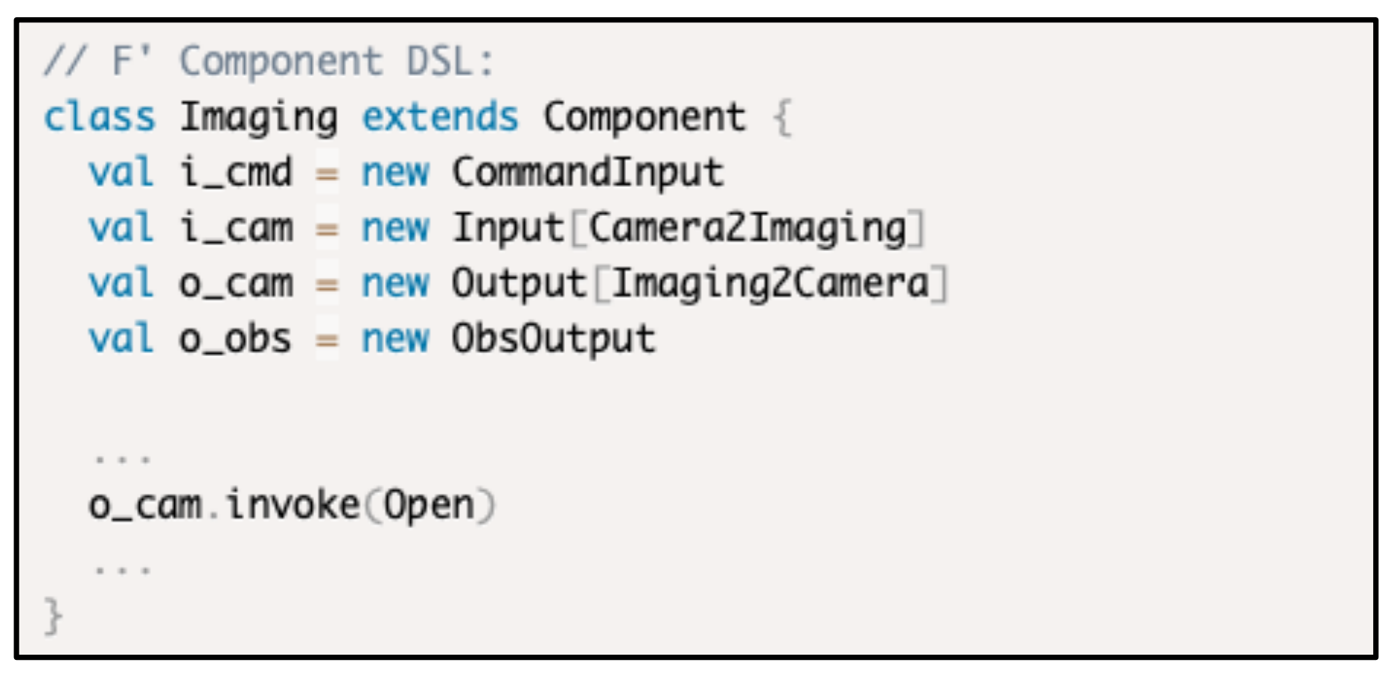
Software (counted in lines of code) on e.g. the Mars rovers grows from mission to mission. Assuming a fixed error count per line of code (1 error per 1000 lines is occasionally used as an estimate), the increasing line count means an increased error count. It is therefore desirable to keep the line count down. This can be done by using more modern programming technology, such as a more modern programming language. The increased focus on autonomy at JPL exacerbates this issue. It is questionable whether JPL can continue coding these complex systems in C. The project will assist JPL in orienting itself towards a new way of writing flight software, resulting in increased programmer productivity and increased reliability.



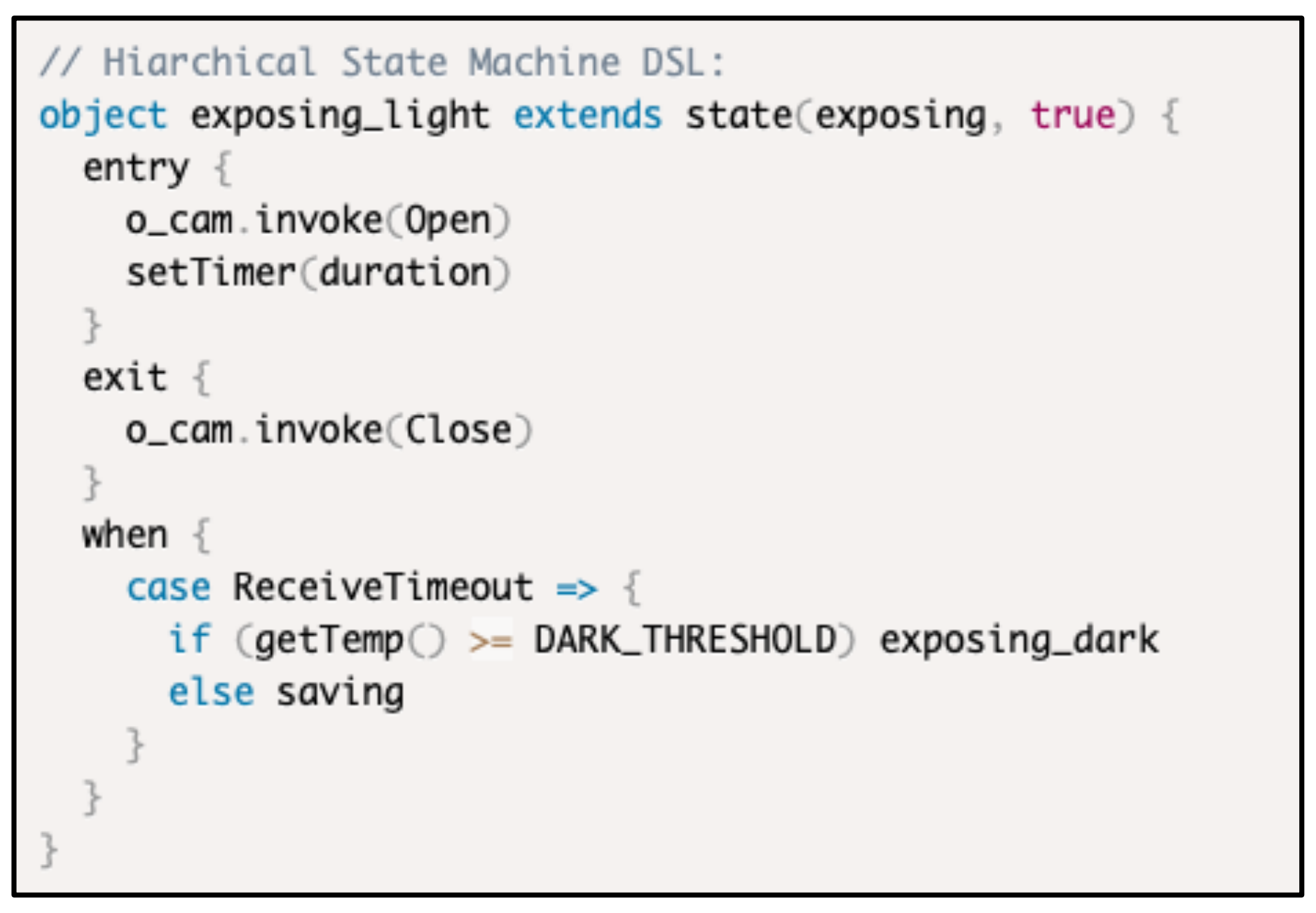
Stay, Buy or Develop	Pros	Cons
Stay with C	easy - no work tested by many low education effort programmers available	language is unsafe weak support for programming in the large programming is low level concurrency not built-in
Buy GPL	relatively easy to do reliance on community tested by many low education effort programmers available	no obvious candidate
Develop new GPL	we have full control easier to develop tools	hard to do no reliance on community not tested by many high education effort no programmers available

FY18/19 Results:

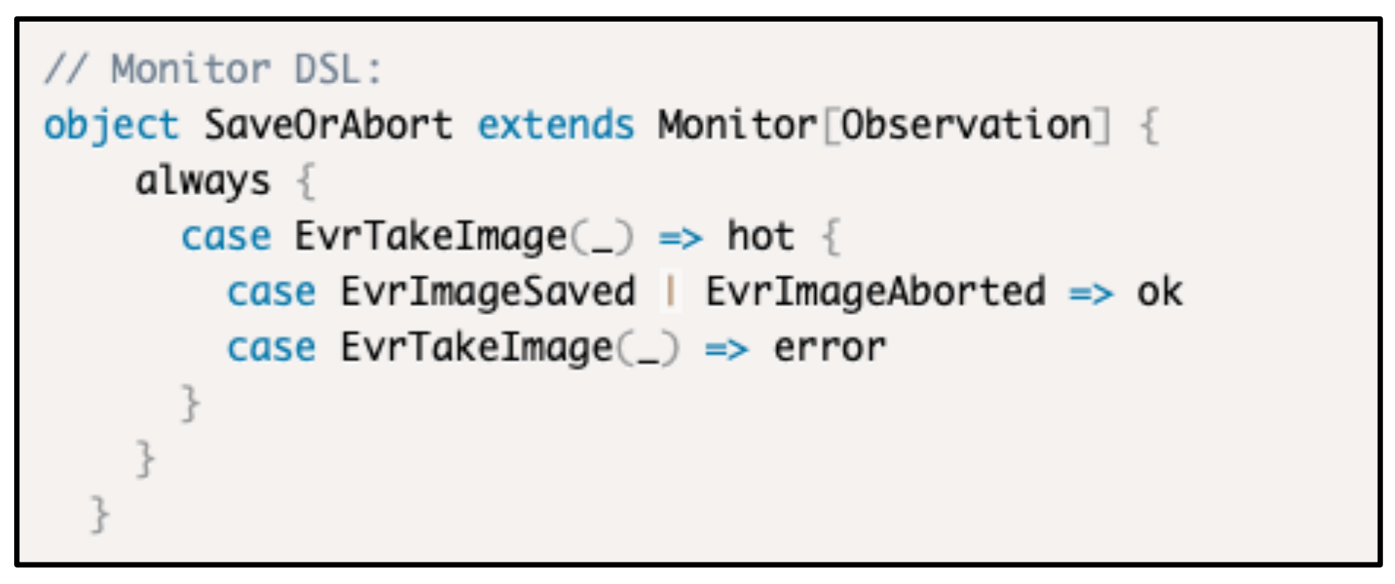
1. We performed a study of the three programming languages Rust, Swift, and Scala as candidates for development of flight software. We performed a study of a selection of frameworks for building domain-specific languages (DSLs), including Racket, Rosette, MPS, and mbeddr. We performed a study of a selection of frameworks for formally verifying programs, including Stainless, Logika, and Viper.
2. We compared the programming languages Rust and Scala on three non-trivial case studies, including (i) the Remote Agent plan execution engine that flew the Deep-Space 1 spacecraft, (ii) a file transmission protocol, and (iii) the F Prime (F') component-based framework (originally written in C++ by the Small Scale Flight Software group 348C).
3. We augmented the F' Scala implementation with three other DSLs for creating respectively (i) hierarchical state machines, (ii) temporal logic monitors, and (iii) non-deterministic rule-based tests.
4. We evaluated, based on the experiments above, future approaches to achieving the objective of lifting programming to a higher abstraction level. Out of the investigated languages, Apple's Swift language looks like the most promising replacement for C for embedded programming, long term.
5. We organized a workshop with the title *'Towards a Unified View of Modeling and Programming'* at ISoLA'18, November 5-6, 2018, Cyprus. Leaders in the fields of modeling and programming were invited to present.



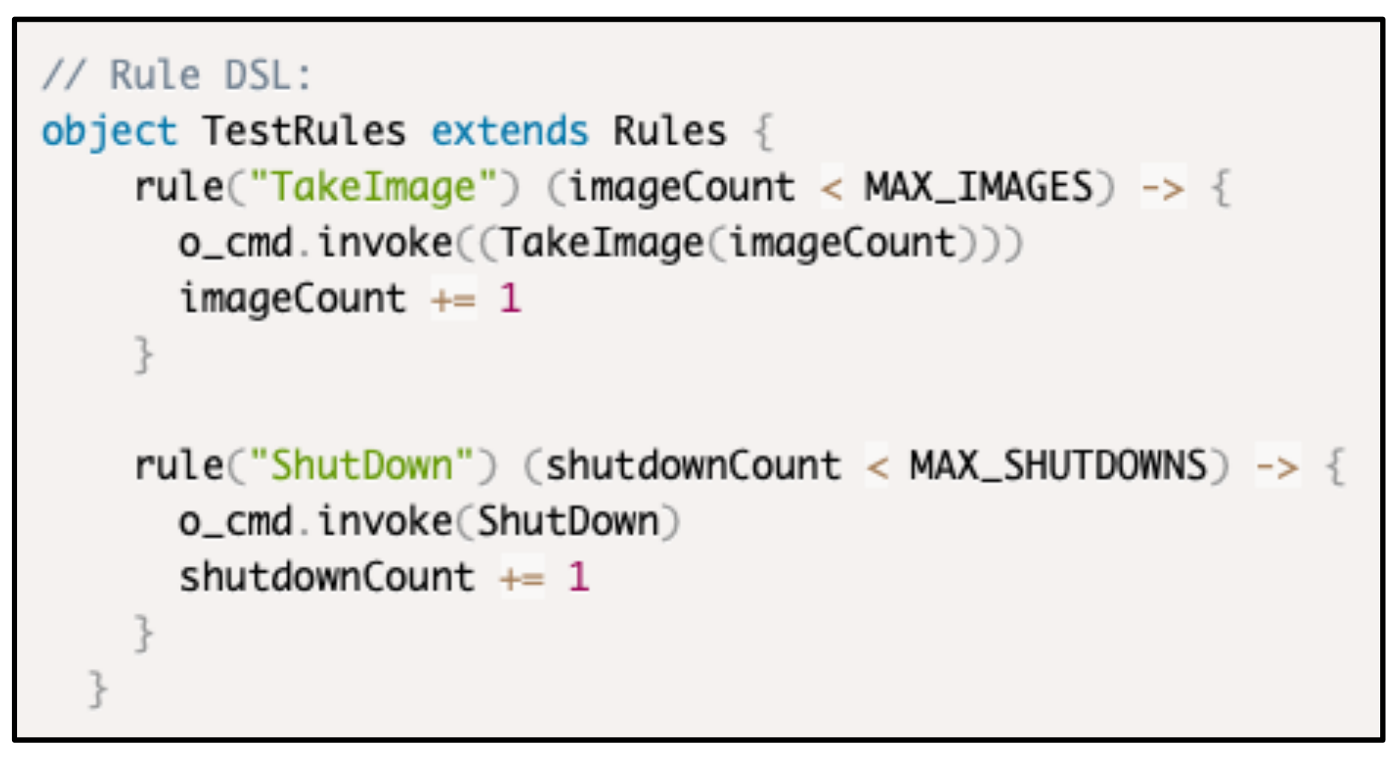
```
// F' Component DSL:
class Imaging extends Component {
  val i_cmd = new CommandInput
  val i_cam = new Input[Camera2Imaging]
  val o_cam = new Output[Imaging2Camera]
  val o_obs = new ObsOutput

  ...
  o_cam.invoke(Open)
  ...
}
```



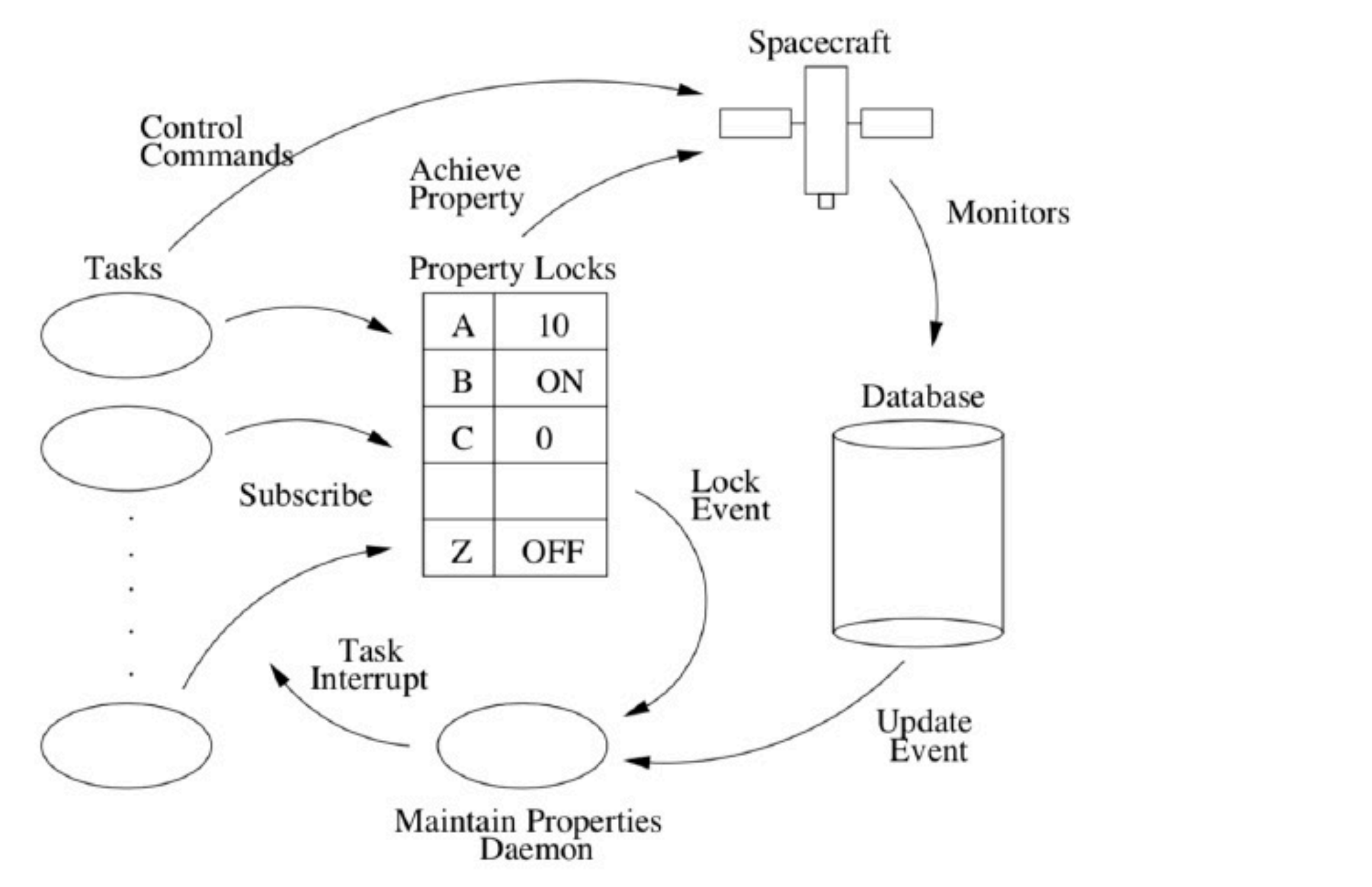
```
// Hiarchical State Machine DSL:
object exposing_light extends state(exposing, true) {
  entry {
    o_cam.invoke(Open)
    setTimer(duration)
  }
  exit {
    o_cam.invoke(Close)
  }
  when {
    case ReceiveTimeout => {
      if (getTemp() >= DARK_THRESHOLD) exposing_dark
      else saving
    }
  }
}
```

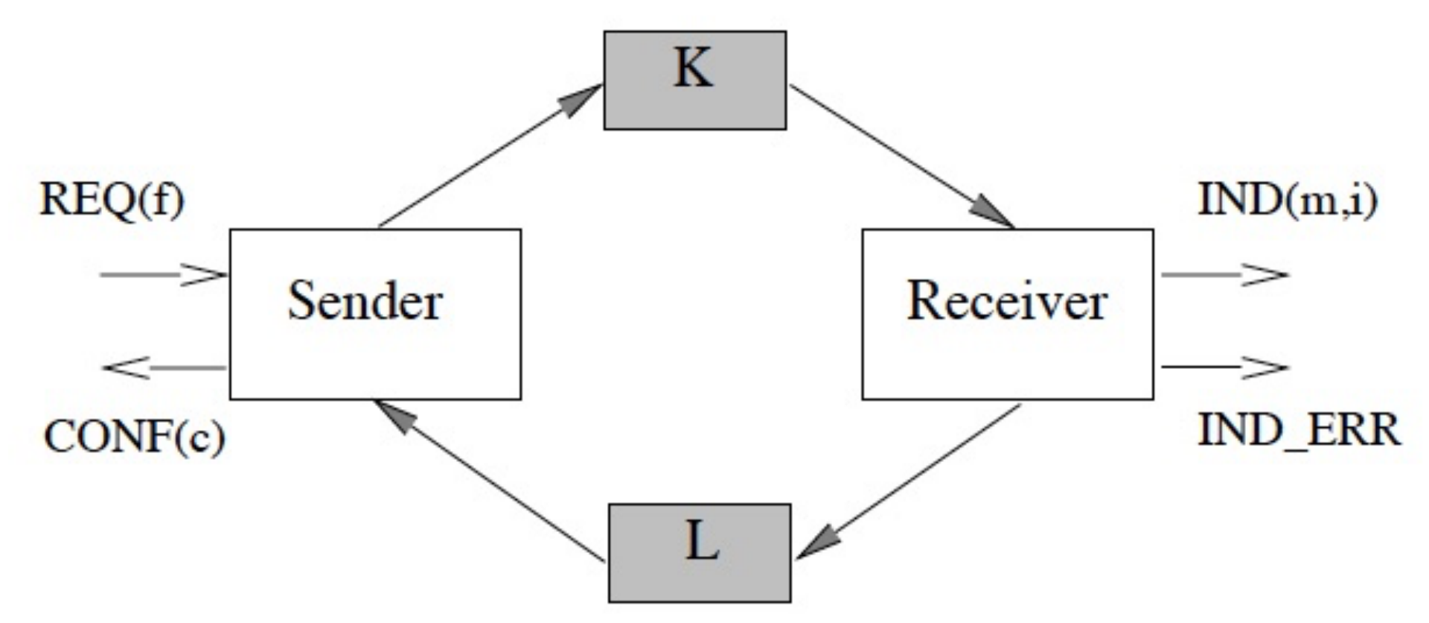


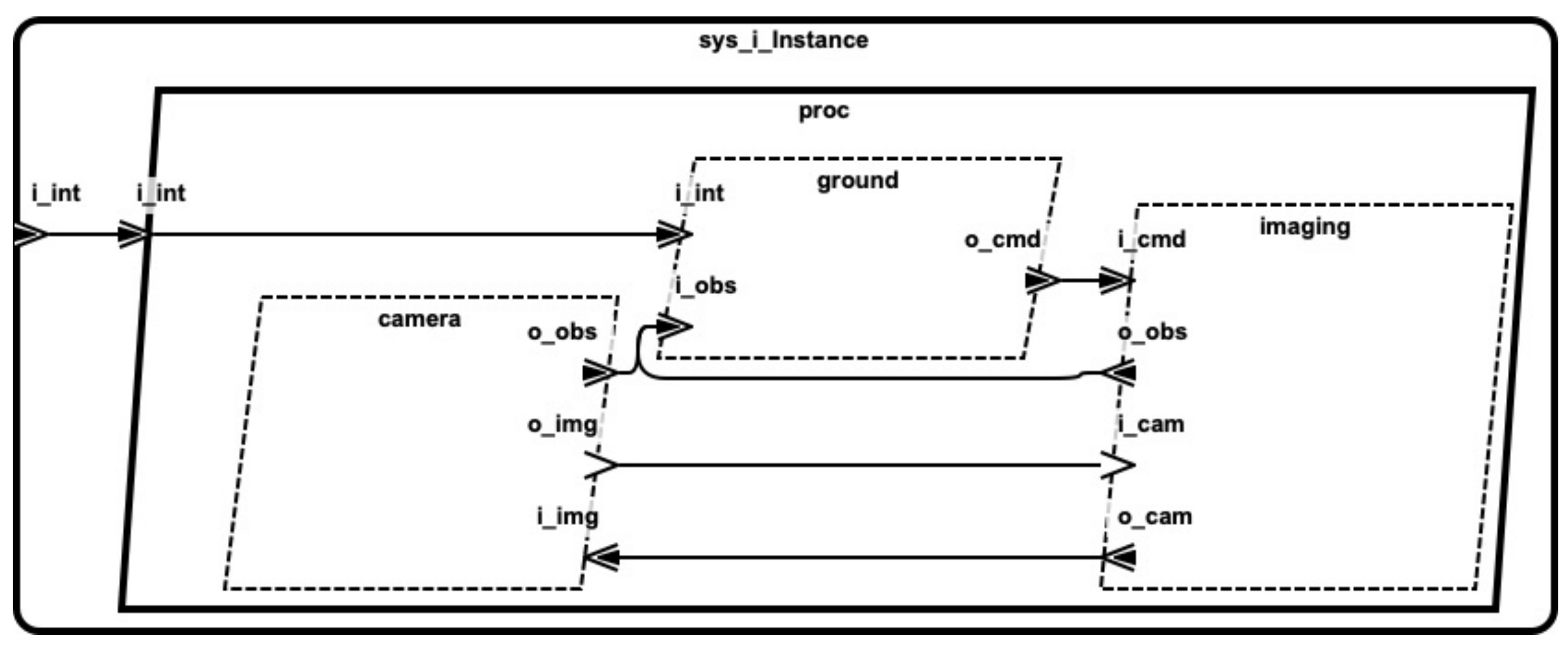
```
// Monitor DSL:
object SaveOrAbort extends Monitor[Observation] {
    always {
      case EvrTakeImage(_) => hot {
        case EvrImageSaved | EvrImageAborted => ok
        case EvrTakeImage(_) => error
      }
    }
  }
```

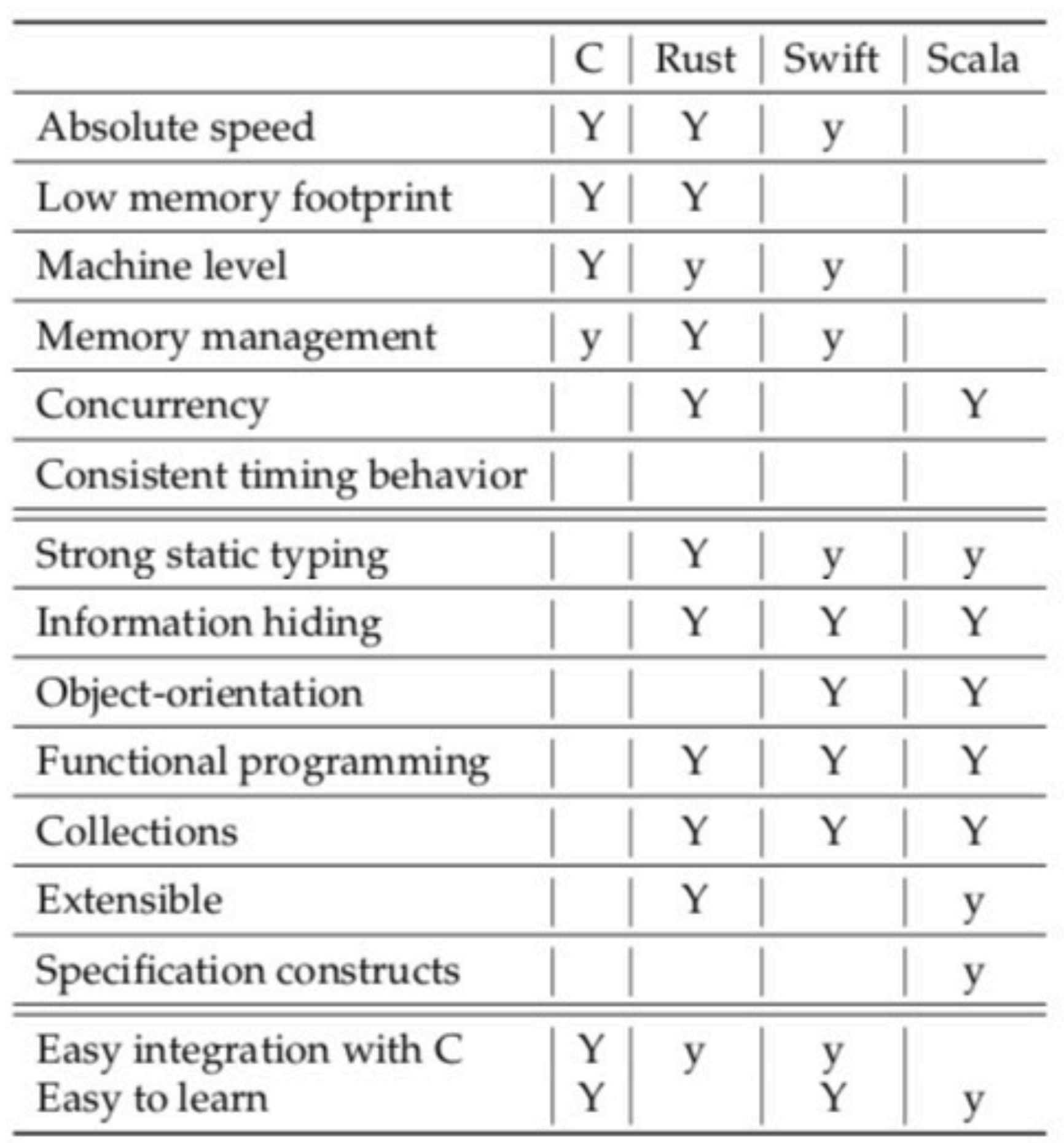


```
// Rule DSL:
object TestRules extends Rules {
    rule("TakeImage") (imageCount < MAX_IMAGES) -> {
      o_cmd.invoke((TakeImage(imageCount)))
      imageCount += 1
    }

    rule("ShutDown") (shutdownCount < MAX_SHUTDOWNS) -> {
      o_cmd.invoke(ShutDown)
      shutdownCount += 1
    }
  }
```









	C	Rust	Swift	Scala
Absolute speed	Y	Y	y	
Low memory footprint	Y	Y		
Machine level	Y	y	y	
Memory management	y	Y	y	
Concurrency		Y		Y
Consistent timing behavior				
Strong static typing		Y	y	y
Information hiding		Y	Y	Y
Object-orientation			Y	Y
Functional programming		Y	Y	Y
Collections		Y	Y	Y
Extensible		Y		y
Specification constructs				y
Easy integration with C	Y	y	y	
Easy to learn	Y		Y	y

Publications:

[1] *Modeling with Scala.*
Klaus Havelund and Rajeev Joshi.
ISoLA 2018, Lecture Notes in Computer Science volume 11244.

[2] *Towards a Unified View of Modeling and Programming.*
Manfred Broy, Klaus Havelund, Rahul Kumar, and Bernhard Steffen.
ISoLA 2018, Lecture Notes in Computer Science volume 11244.

PI/Task Mgr. Contact Information:

Phone : 818-354-5418
Email : klaus.havelund@jpl.nasa.gov

National Aeronautics and Space Administration

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

www.nasa.gov

Poster No. RPC-131