The Art of Building Tools

A Language Engineering Perspective

Markus Voelter
independent/itemis

http://voelter.de
voelter@acm.org
@markusvoelter

1 Tool Extension

2 Ex: mbeddr

3 GTSL

4 Ex: Requirements

5 Ex: Insurance

6 Wrap Up

1

Tool Extension

Tool Extensibility

Study Findings I

The majority of our interviewees were very successful with MDE but all of them either **built their own** modeling tools, made **heavy adaptations** of off-the-shelf tools, or spent a lot of time finding ways to **work around** tools. The only accounts of easy-to-use, intuitive tools came from those who had developed tools themselves for bespoke purposes. Indeed, this suggests that current tools are a barrier to success rather than an enabler.

Jon Whittle, John Hutchinson, Mark Rouncefield, Hakan Burden, and Rogardt Heldal. Industrial Adoption of Model-Driven Engineering: Are the Tools Really the Problem? In *Proceedings of the 16th International Conference on Model Driven Engineering Languages and Systems (MODELS) 2013*. ACM, 2013.

Tool Extensibility

Study Findings II

Complexity problems are typically associated with off-the- shelf tools. Of particular note is **accidental complexity** – which can be introduced due to [..] [the] lack of flexibility to adapt the tools to a company's own context [..]

Jon Whittle, John Hutchinson, Mark Rouncefield, Hakan Burden, and Rogardt Heldal. Industrial Adoption of Model-Driven Engineering: Are the Tools Really the Problem? In *Proceedings of the 16th International Conference on Model Driven Engineering Languages and Systems (MODELS) 2013*. ACM, 2013.

Tool Extensibility

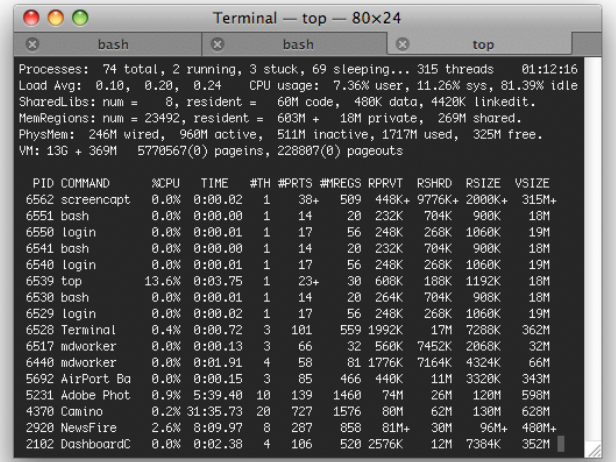
Study Findings III

Our interviews point to a **strong need for tailoring** of some sort: either tailor the tool to the process, tailor the process to the tool, or build your own tool that naturally fits your own process. Based on our data, it seems that, on balance, it is currently much easier to do the latter.

Jon Whittle, John Hutchinson, Mark Rouncefield, Hakan Burden, and Rogardt Heldal. Industrial Adoption of Model-Driven Engineering: Are the Tools Really the Problem? In *Proceedings of the 16th International Conference on Model Driven Engineering Languages and Systems (MODELS) 2013*. ACM, 2013.

Tool Extensibility

Command-Line Tools



```
Processes:  74 total, 2 running, 3 stuck, 69 sleeping... 315 threads    01:12:16
Load Avg:  0.10,  0.20,  0.24    CPU usage:  7.36% user, 11.26% sys, 81.39% idle
SharedLibs: num =    8, resident =   60M code,  480K data, 4420K linkedit.
MemRegions: num = 23492, resident =  603M +   18M private,  269M shared.
PhysMem:  246M wired,  960M active,  511M inactive, 1717M used,  325M free.
VM: 13G + 369M   5770567(0) pageins, 228807(0) pageouts

  PID COMMAND      %CPU   TIME   #TH #PRTS #MREGS RPRVT  RSHRD  RSIZE  VSIZE
 6562 screencapt   0.0%  0:00.02   1    38+    509  448K+ 9776K+ 2000K+  315M+
 6551 bash         0.0%  0:00.00   1    14     20  232K   704K   900K    18M
 6550 login        0.0%  0:00.01   1    17     56  248K   268K  1060K    19M
 6541 bash         0.0%  0:00.00   1    14     20  232K   704K   900K    18M
 6540 login        0.0%  0:00.01   1    17     56  248K   268K  1060K    19M
 6539 top         13.6%  0:03.75   1    23+    30  608K   188K  1192K    18M
 6530 bash         0.0%  0:00.01   1    14     20  264K   704K   908K    18M
 6529 login        0.0%  0:00.02   1    17     56  248K   268K  1060K    19M
 6528 Terminal     0.4%  0:00.72   3   101    559 1992K    17M  7288K   362M
 6517 mdworker     0.0%  0:00.13   3    66     32  560K  7452K  2068K    32M
 6440 mdworker     0.0%  0:01.91   4    58     81 1776K  7164K  4324K    66M
 5692 AirPort Ba   0.0%  0:00.15   3    85    466  440K    11M  3320K   343M
 5231 Adobe Phot   0.9%  5:39.40  10   139   1460   74M    26M   120M   598M
 4370 Camino       0.2% 31:35.73  20   727   1576   80M    62M   130M   628M
 2920 NewsFire     2.6%  8:09.97   8   287    858   81M+   30M    96M+  480M+
 2102 DashboardC   0.0%  0:02.38   4   106    520 2576K    12M  7384K   352M
```

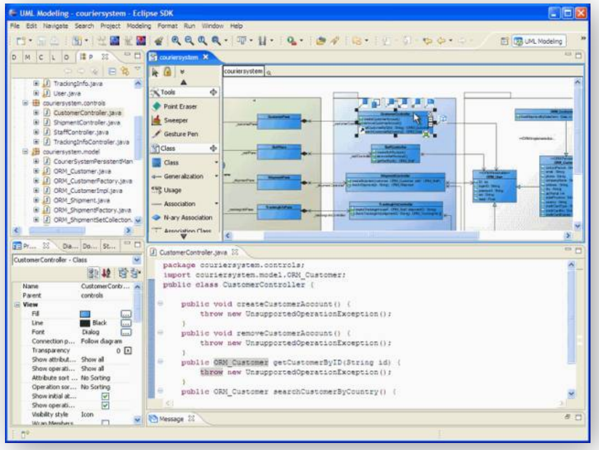
New File Formats

New Processors

Assemble Components (Pipes & Filters)

Tool Extensibility

UI Tools



```
package couriersystem.controls;
import couriersystem.model.ORM_Customer;
public class CustomerController {

    public void createCustomerAccount() {
        throw new UnsupportedOperationException();
    }
    public void removeCustomerAccount() {
        throw new UnsupportedOperationException();
    }
    public ORM_Customer getCustomerByID(String id) {
        throw new UnsupportedOperationException();
    }
    public ORM_Customer searchCustomerByCountry() {
```

Buttons Views

Menus Actions

(New Languages)

(New Editors)

Platform/Plugin Systems

2

mbeddr

An Example System

Language Engineering Embedded Software

mbeddr

An Example System

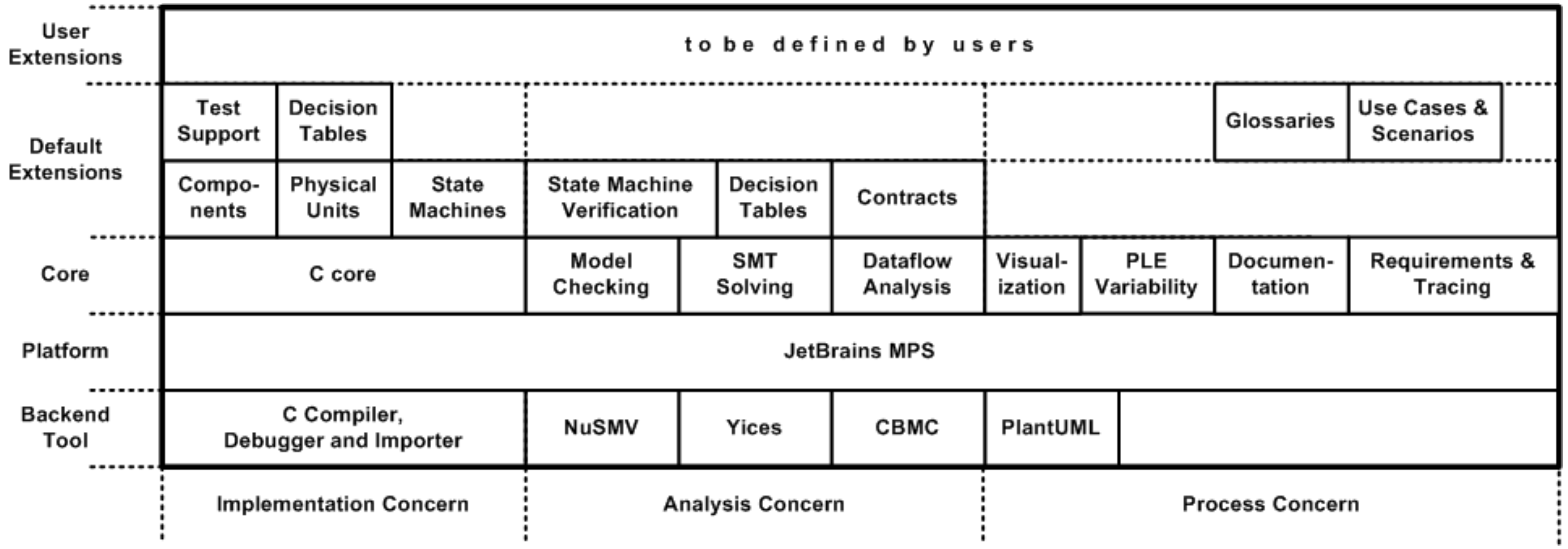
Language Engineering Embedded Software

A collection of integrated languages for embedded software engineering.

An Example System

Language Engineering Embedded Software

An extensible collection of integrated languages for embedded software engineering.



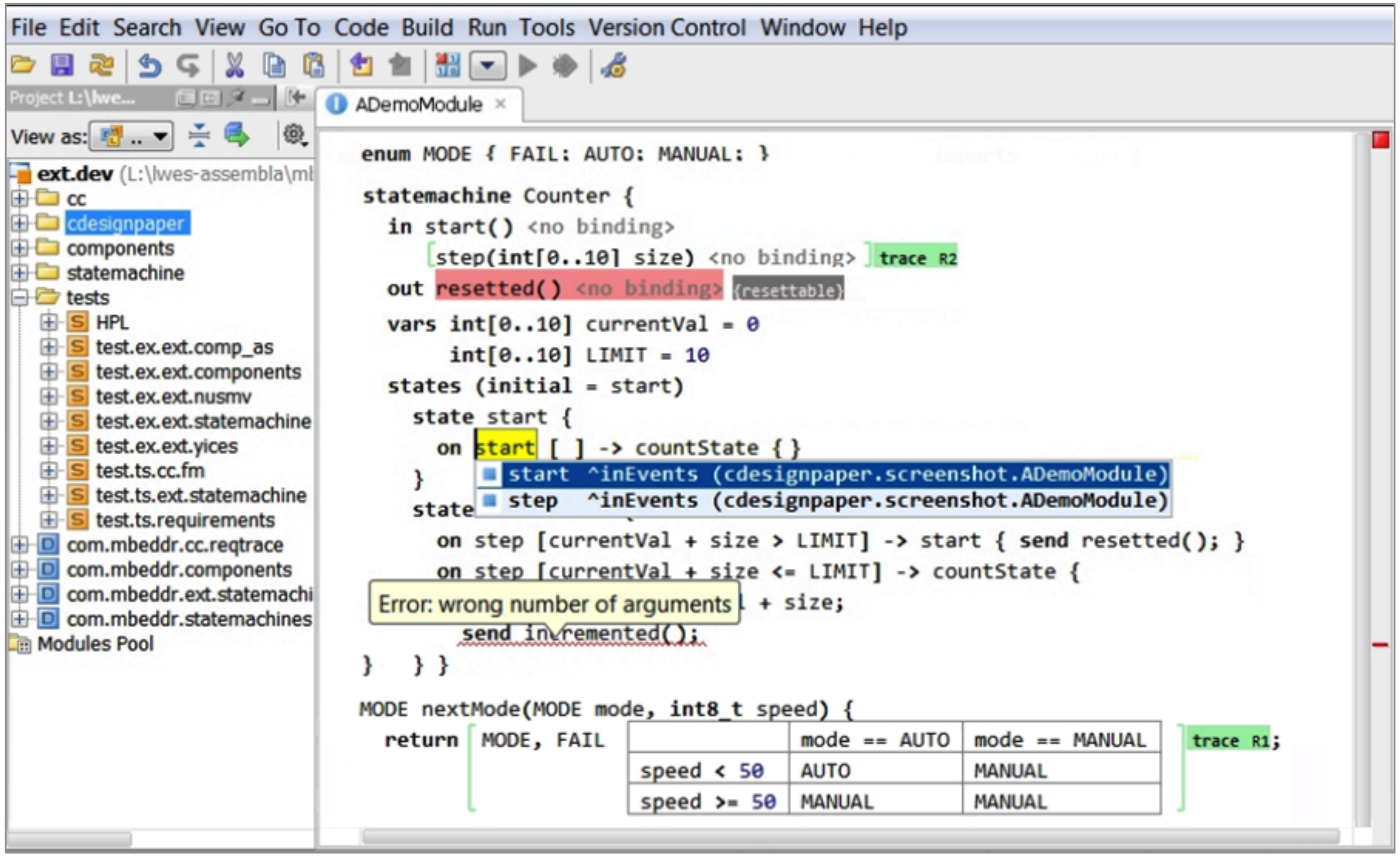
Specific Languages

An Example System

mbeddr

Language Engineering Embedded Software

An IDE for all Of them



```
enum MODE { FAIL: AUTO: MANUAL: }

statemachine Counter {
  in start() <no binding>
     [step(int[0..10] size) <no binding> ] trace R2
  out resetted() <no binding> {resettable}
  vars int[0..10] currentVal = 0
       int[0..10] LIMIT = 10
  states (initial = start)
    state start {
      on start [ ] -> countState { }
    }
    state
      on step [currentVal + size > LIMIT] -> start { send resetted(); }
      on step [currentVal + size <= LIMIT] -> countState {
                                            l + size;
        send incremented();
}   } }

MODE nextMode(MODE mode, int8_t speed) {
  return
```

start ^inEvents (cdesignpaper.screenshot.ADemoModule)
step ^inEvents (cdesignpaper.screenshot.ADemoModule)

Error: wrong number of arguments

return [MODE, FAIL

	mode == AUTO	mode == MANUAL
speed < 50	AUTO	MANUAL
speed >= 50	MANUAL	MANUAL

] trace R1;

Specific Languages

An Example System

Language Engineering Embedded Software

Open Source
Eclipse Public License

http://mbeddr.com

itemis fortiss BMW Car IT Bundesministerium für Bildung und Forschung

Specific Languages

About mbeddr

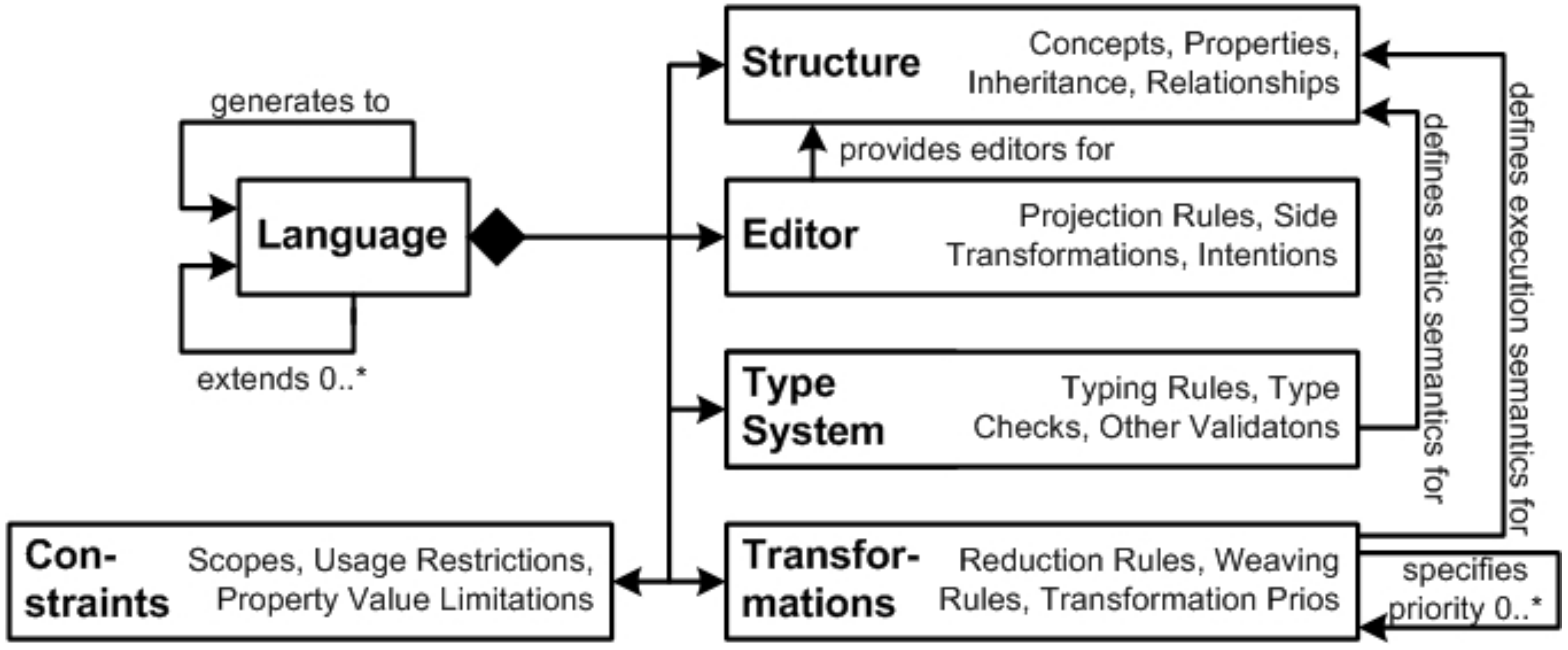
Built on JetBrains MPS

Open Source
Apache 2.0
http://jetbrains.com/mps

Generic Tool

About MPS

Rich Set of Language Aspects



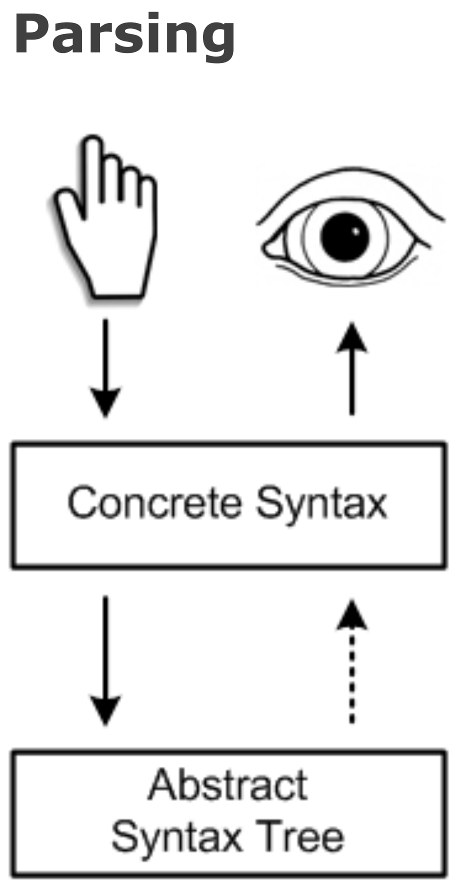
generates to
Language
extends 0..*
Structure
Concepts, Properties, Inheritance, Relationships
provides editors for
Editor
Projection Rules, Side Transformations, Intentions
Type System
Typing Rules, Type Checks, Other Validatons
Con-straints
Scopes, Usage Restrictions, Property Value Limitations
Transfor-mations
Reduction Rules, Weaving Rules, Transformation Prios
specifies priority 0..*
defines static semantics for
defines execution semantics for

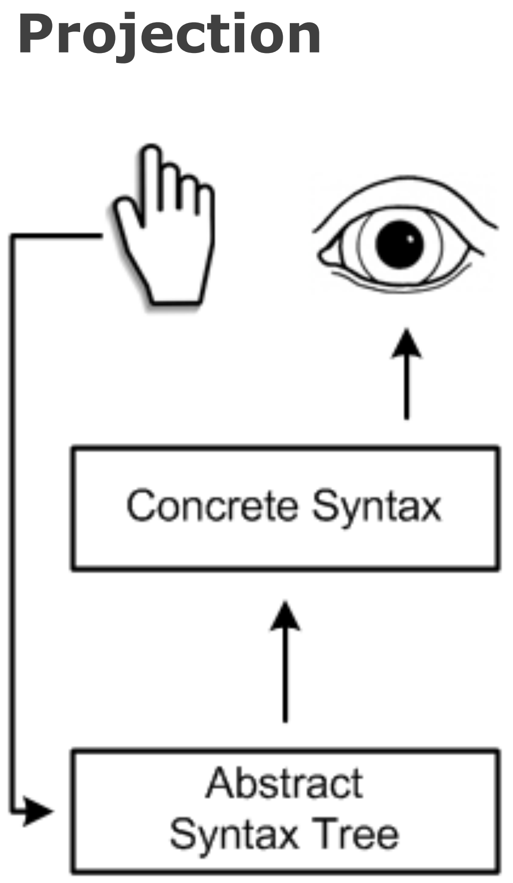
+ Refactorings, Find Usages, Syntax Coloring, Debugging, ...

Generic Tool

About MPS

Projectional Editing





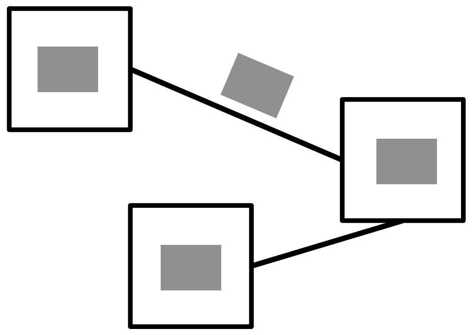
Notational Flexibility

Regular Code/Text

Mathematical

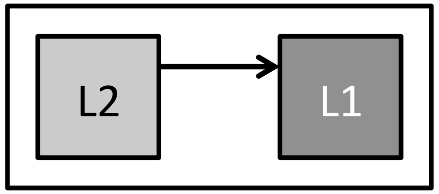
Tables

Graphical



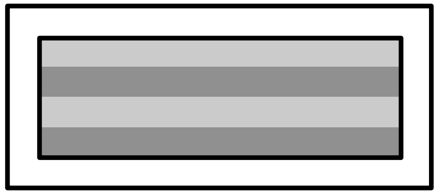
About MPS

Language Composition



Separate Files

Type System
Transformation
Constraints

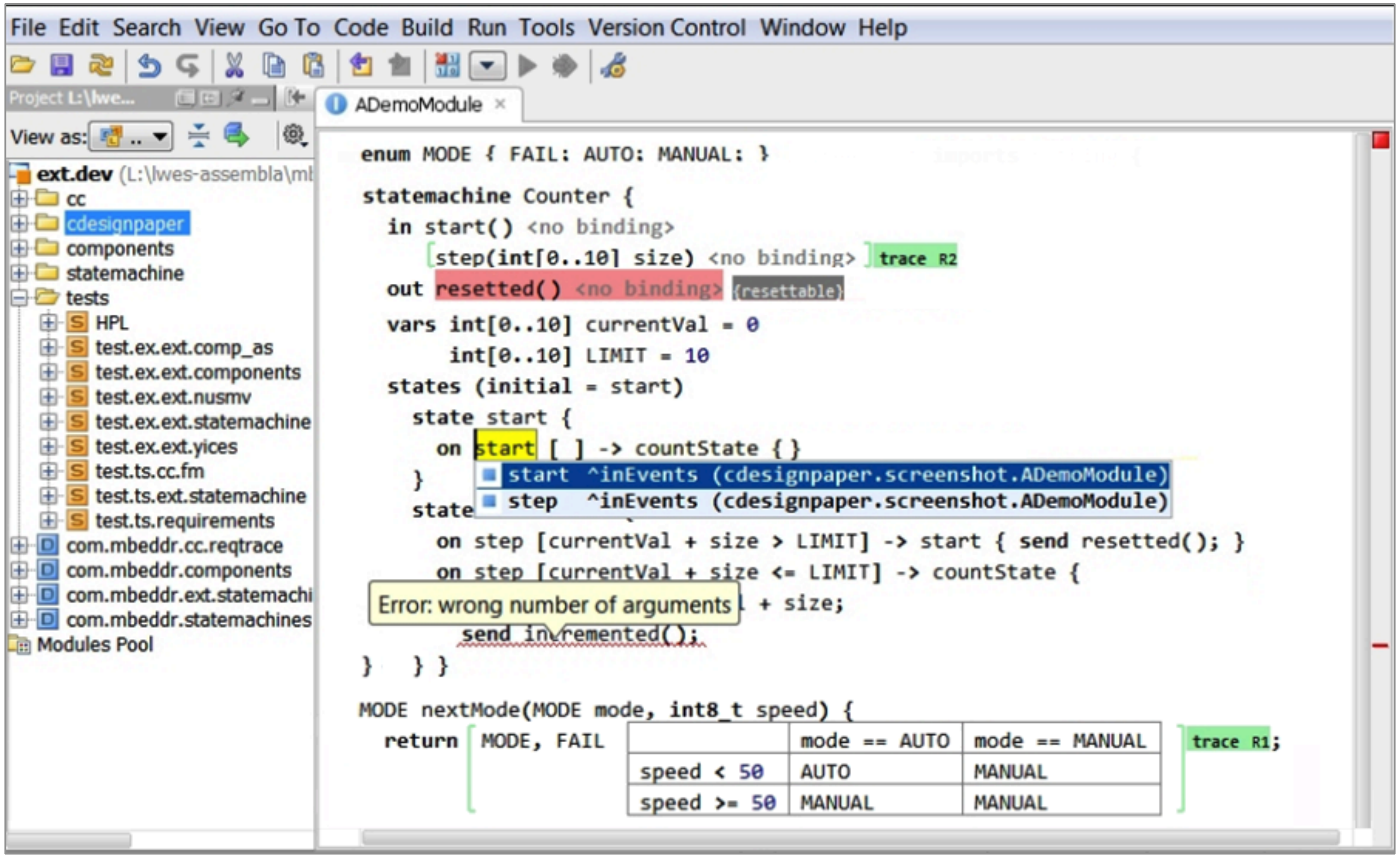


In One File

Type System
Transformation
Constraints
Syntax
Editor/IDE

An Example System

Built on JetBrains MPS



```
enum MODE { FAIL: AUTO: MANUAL: }

statemachine Counter {
  in start() <no binding>
     step(int[0..10] size) <no binding>   trace R2
  out resetted() <no binding> {resettable}
  vars int[0..10] currentVal = 0
       int[0..10] LIMIT = 10
  states (initial = start)
    state start {
      on start [ ] -> countState { }
         start  ^inEvents (cdesignpaper.screenshot.ADemoModule)
         step   ^inEvents (cdesignpaper.screenshot.ADemoModule)
    }
    state
      on step [currentVal + size > LIMIT] -> start { send resetted(); }
      on step [currentVal + size <= LIMIT] -> countState {
 Error: wrong number of arguments  + size;
        send incremented();
}   } }

MODE nextMode(MODE mode, int8_t speed) {
  return  MODE, FAIL
```

	mode == AUTO	mode == MANUAL
speed < 50	AUTO	MANUAL
speed >= 50	MANUAL	MANUAL

trace R1;

Generic Tool

Features

Hello, World

```
module HelloWorld {
  messagelist messages {
    INFO helloWorld() active: Hello, World
  }
  exported int32 main(int32 argc, string*[] argv) {
    report(0) messages.helloWorld();
    return 0;
  }
}
```

Messages abstract over IO. Report statements output them.

Features

Function Types and Function Pointers

```
typedef (Trackpoint*)=>(Trackpoint*) as DataProcessorType;
DataProcessorType processor;

Trackpoint process_nullifyAlt(Trackpoint* tp) {
  tp->alt = 0;
  return e;
}

test case testProcessing {
  Trackpoint tp = {x = 0, y = 0, alt = 100 };
  processor = :process_nullifyAlt;
  Trackpoint* res = processor(&tp);
  assert(1) res->alt == 0;
}
```

Better notation for function types and function references.

Features

Function Types and Lambdas

```
typedef (Trackpoint*)=>(Trackpoint*) as DataProcessorType;
DataProcessorType processor;

Trackpoint process_nullifyAlt(Trackpoint* tp) {
  tp->alt = 0;
  return e;
}

test case testLambdaProcessing {
  Trackpoint tp = {x = 0, y = 0, alt = 50 };
  processor = [p| p->alt = 100; p; ];
  assert(0) processor(tp)->alt == 100;
}
```

And yes, we have lambdas (it's 2013 after all ☺)

Features

Reporting

```
void addToQueue(Trackpoint* tp) {
  report(0) messages.queueGettingFull()
            on pos >= QUEUE_SIZE * 3/4;
  pos++;
  if ( pos >= QUEUE_SIZE ) pos = 0;
  queue[pos] = tp;
}
```

Reporting has conditions. All of it removed, when disabled!

Features

Test Cases

```
test case testProcessing {
  Trackpoint tp = {x = 0, y = 0, alt = 100 };
  processor = :process_nullifyAlt;
  Trackpoint* res = processor(&tp);
  assert(1) res->alt == 0;
}

exported int32 main(int32 argc, string[] argv) {
  return test testAddToQueue, testQueueFilling;
}
```

Special expression to run test cases and collect failure count.

Features

mbeddr

Physical Units

```
struct Trackpoint {
  int8 id;                  // sequence ID of the trackpoint
  int8/s/   timestamp; // timestamp as taken from GPS time
  int8/m/   x;          // longtitude, simplified as a number
  int8/m/   y;          // latitude, simplified as a number
  int8/m/   alt;        // altitude as of the GPS
  int8/mps/ speed;      // current speed, if available
};
```

derived unit mps = $m\ s^{-1}$ **for** velocity

Types can have units; additional units can be defined.

Features

Physical Units II

```
Trackpoint tp = { id = 1, timestamp = 0 s,
                  x = 0 m, y = 0 m, alt = 100 m };
assert(0) tp.id == 1 && tp.alt == 100 m;
assert(1) tp.id == 1 && tp.alt == 0 m;


int8 someInt = tp.x + tp.speed; // error, adding m and mps


int8/mps/ speed = (tp2.x - tp1.x) /
                  (tp2.timestamp - tp1.timestamp);
```

Literals can have units; type system calculates w/ units.

Features

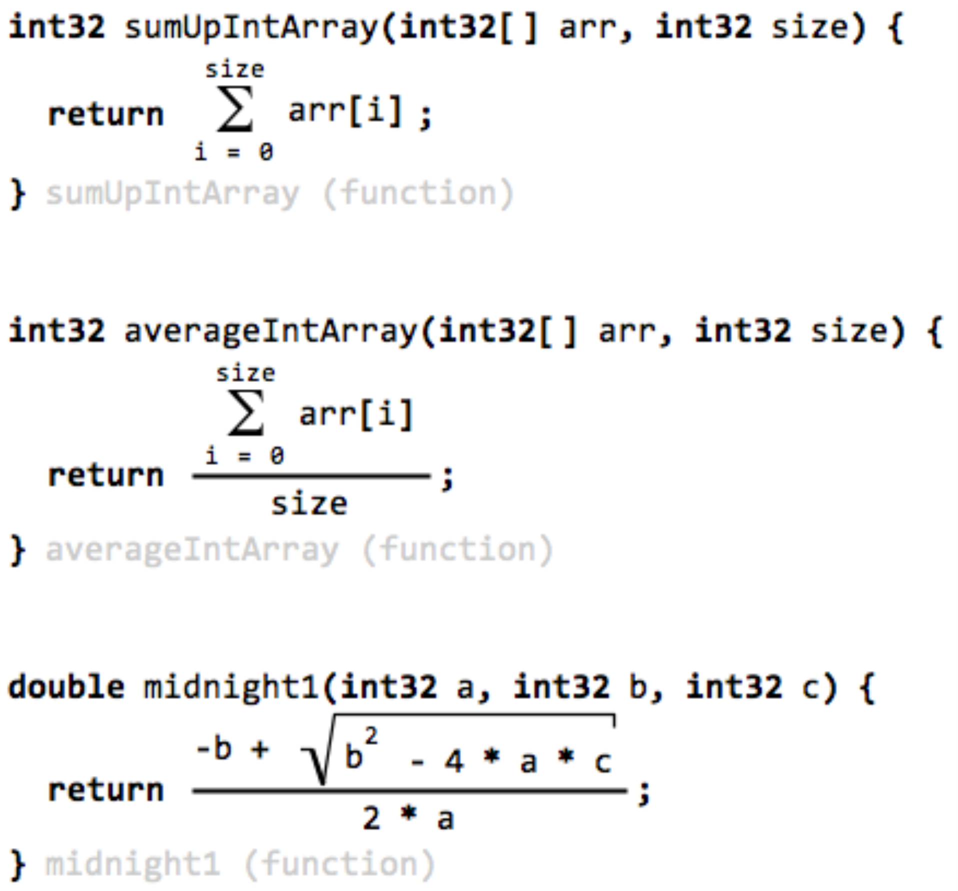
Physical Units III

```
convertible unit degC for temperature
convertible unit degF for temperature
conversion degC -> degF = val * 9 / 5 + 32
conversion degF -> degC = (val - 32) * 5 / 9

void storeTemperature(int8/degC/ temp) {
  // store temp in some data store
}


int8/degF/ aTempInF = 100 degF;
storeTemperature(convert[aTempInF -> degC]);
```

*Convertible units support *value* conversions!*

Features

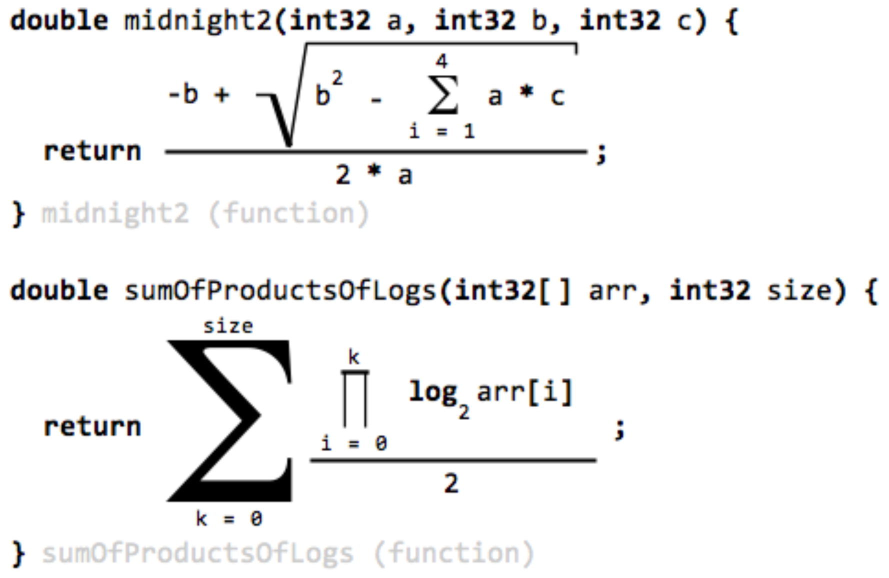
Math



```
int32 sumUpIntArray(int32[] arr, int32 size) {
```
$$\text{return } \sum_{i = 0}^{size} arr[i] ;$$
```
} sumUpIntArray (function)
```

```
int32 averageIntArray(int32[] arr, int32 size) {
```
$$\text{return } \frac{\sum_{i = 0}^{size} arr[i]}{size} ;$$
```
} averageIntArray (function)
```

```
double midnight1(int32 a, int32 b, int32 c) {
```
$$\text{return } \frac{-b + \sqrt{b^{2} - 4 * a * c}}{2 * a} ;$$
```
} midnight1 (function)
```



```
double midnight2(int32 a, int32 b, int32 c) {
```
$$\text{return } \frac{-b + \sqrt{b^{2} - \sum_{i = 1}^{4} a * c}}{2 * a} ;$$
```
} midnight2 (function)
```

```
double sumOfProductsOfLogs(int32[] arr, int32 size) {
```
$$\text{return } \sum_{k = 0}^{size} \frac{\prod_{i = 0}^{k} \log_{2} arr[i]}{2} ;$$
```
} sumOfProductsOfLogs (function)
```

Support for readable mathematical symbols.

Features

Interfaces and Components I

```
module Components imports DataStructures {
  exported cs interface TrackpointProcessor {
    Trackpoint* process(Trackpoint* p);
  }
}

exported component Nuller extends nothing {
  provides TrackpointProcessor processor
  Trackpoint* process(Trackpoint* p) <- op processor.process {
    p->alt = 0 m;
    return p;
  }
}
```

Interfaces define operations. Components provide interfaces.

Features

Interfaces and Components II

```
instances nullerInstances {
  instance Nuller nuller
  adapt n -> nuller.processor
}

exported test case testNuller {
  initialize nullerInstances;
  Trackpoint tp = { id = 0 };
  n.process(&tp);
}
```

Components can be instantiated and wired.

Features

Interfaces and Components III

```
exported cs interface TrackpointStore1 {
  void store(Trackpoint* tp)
    pre(0) isEmpty()
    pre(1) tp != null
    post(2) !isEmpty()
    post(3) size() == old(size()) + 1
  Trackpoint* get()
    pre(0) !isEmpty()
  Trackpoint* take()
    pre(0) !isEmpty()
    post(1) result != null
    post(2) isEmpty()
    post(3) size() == old(size()) - 1
  query int8 size()
  query boolean isEmpty()
}
```

Interfaces can have pre- and postconditions.

Features

Interfaces and Components IV

```
exported cs interface TrackpointStore2 {
  // store goes from the initial state to a new state nonEmpty
  void store(Trackpoint* tp)
    protocol init(0) -> new nonEmpty(1)

  // get expects the state to be nonEmpty, and remains there
  Trackpoint* get()
    protocol nonEmpty -> nonEmpty

  // take expects to be nonEmpty and then becomes empty
  // if there was one element in it, it remains in
  // nonEmpty otherwise
  Trackpoint* take()
    post(0) result != null
    protocol nonEmpty [size() == 1] -> init(0)
    protocol nonEmpty [size() > 1] -> nonEmpty

  // isEmpty and size have no effect on the protocol state
  query boolean isEmpty()
  query int8 size()
}
```

In addition, interfaces can have protocol state machines.

Features

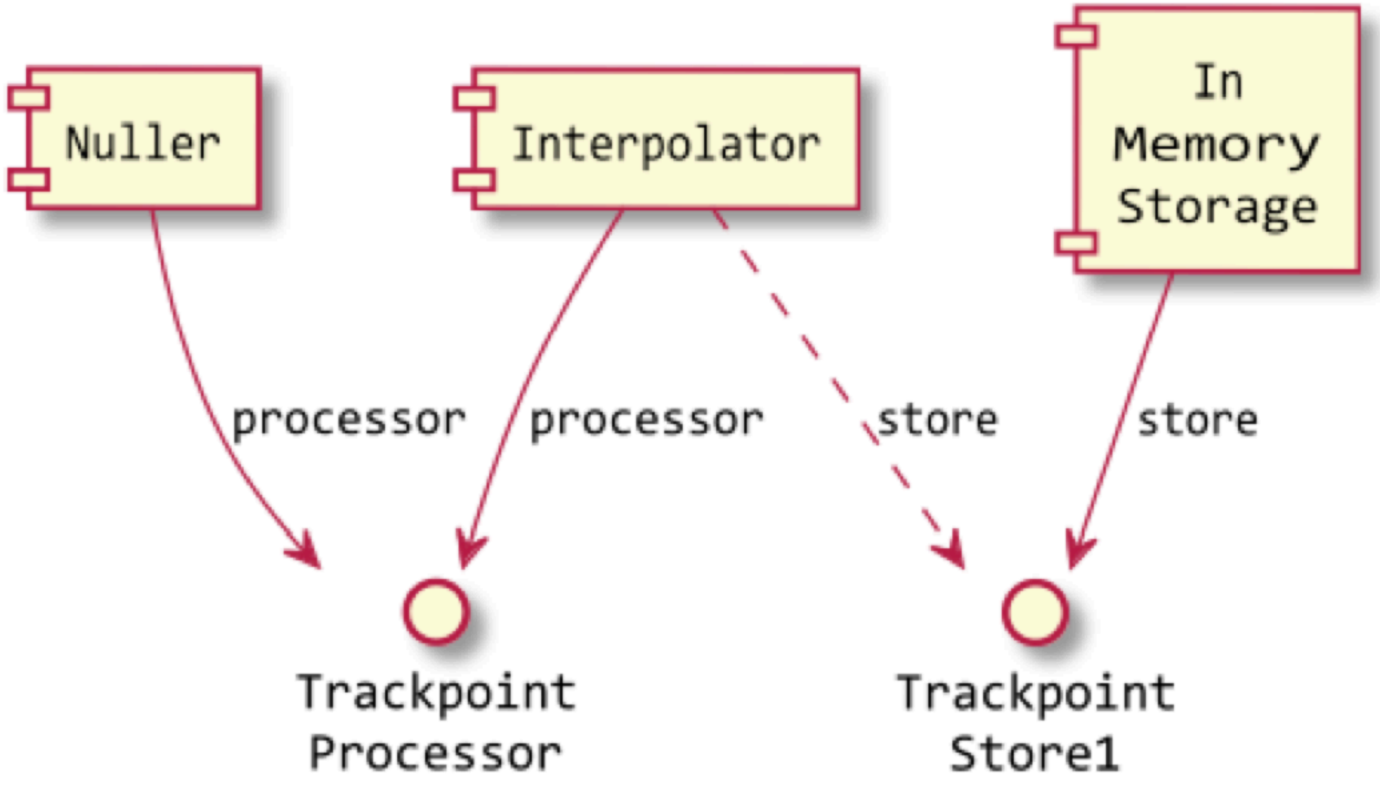
Interfaces and Components V

```
exported component Interpolator extends nothing {
  provides TrackpointProcessor processor
  requires TrackpointStore store

  init int8 divident;
  Trackpoint* process(Trackpoint* p) <- op processor.process {
    if (store.isEmpty()) {
      store.store(p);
      return p;
    } else {
      Trackpoint* old = store.take();
      p->speed = (p->speed + old->speed) / divident;
      store.store(p);
      return p;
    }
  }
}
```

Components can also require ports (dependency injection)

Features

Interfaces and Components VI



Interfaces and components can be visualized.

Interfaces and Components VIII

```
mock component StorageMock report messages: true {
  provides TrackpointStore1 store
  Trackpoint* lastTP;
  total no. of calls is 5
  sequence {
    step 0: store.isEmpty return true;
    step 1: store.store {
        assert 0: parameter tp: tp != null
      }
      do { lastTP = tp; }
    step 2: store.isEmpty return false;
    step 3: store.take return lastTP;
    step 4: store.store
  }
}
```

Mock components specify expectations in context of a test.

Features

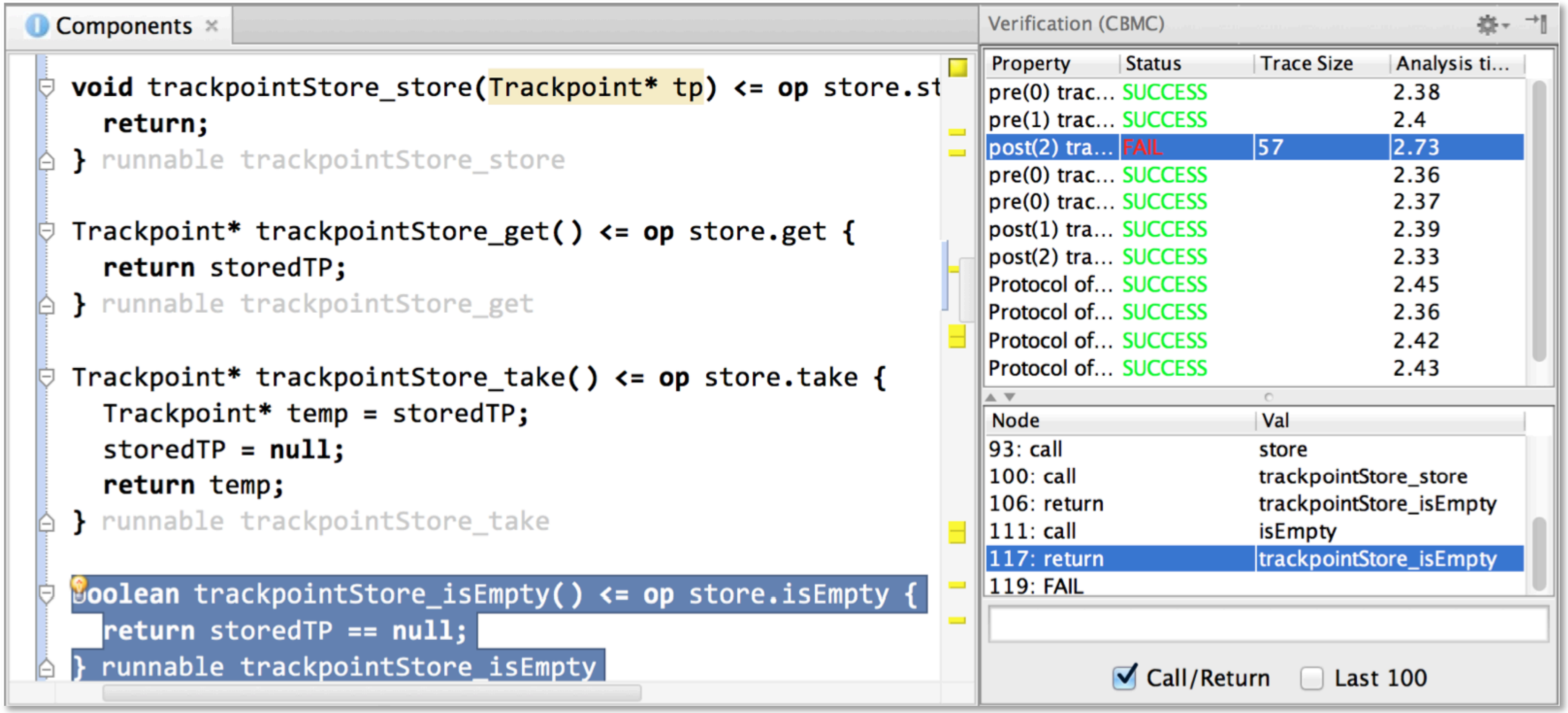
Interfaces and Components IX

```
exported test case testInterpolatorWithMock {
  initialize interpolatorInstancesWithMock;
  Trackpoint p1 = { id = 1, timestamp = 1 s, speed = 10 mps };
  Trackpoint p2 = { id = 2, timestamp = 2 s, speed = 20 mps };
  ipMock.process(&p1);
  ipMock.process(&p2);
  validatemock (0) interpolatorInstancesWithMock:storeMock;
}
```

Mocks can be instantiated and validated in tests.

Features

Interfaces and Components X



```
void trackpointStore_store(Trackpoint* tp) <= op store.st
  return;
} runnable trackpointStore_store

Trackpoint* trackpointStore_get() <= op store.get {
  return storedTP;
} runnable trackpointStore_get

Trackpoint* trackpointStore_take() <= op store.take {
  Trackpoint* temp = storedTP;
  storedTP = null;
  return temp;
} runnable trackpointStore_take

boolean trackpointStore_isEmpty() <= op store.isEmpty {
  return storedTP == null;
} runnable trackpointStore_isEmpty
```

Property	Status	Trace Size	Analysis ti...
pre(0) trac...	SUCCESS		2.38
pre(1) trac...	SUCCESS		2.4
post(2) tra...	FAIL	57	2.73
pre(0) trac...	SUCCESS		2.36
pre(0) trac...	SUCCESS		2.37
post(1) tra...	SUCCESS		2.39
post(2) tra...	SUCCESS		2.33
Protocol of...	SUCCESS		2.45
Protocol of...	SUCCESS		2.36
Protocol of...	SUCCESS		2.42
Protocol of...	SUCCESS		2.43

Node	Val
93: call	store
100: call	trackpointStore_store
106: return	trackpointStore_isEmpty
111: call	isEmpty
117: return	trackpointStore_isEmpty
119: FAIL	

[✓] Call/Return ☐ Last 100

Interface contracts can be verified statically!

Features

Decision Tables

```
exported component Judge extends nothing {
  provides FlightJudger judger
  int16 points = 0;
  void judger_reset() <= op judger.reset {
    points = 0;
  } runnable judger_reset
  void judger_addTrackpoint(Trackpoint* tp) <= op judger.addTrackpoint {
    points += 0
```

	tp->alt <= 2000 m	tp->alt >= 2000 m
tp->speed < 150 mps	0	10
tp->speed >= 150 mps	5	20

```
  } runnable judger_addTrackpoint
  int16 judger_getResult() <= op judger.getResult {
    return points;
  } runnable judger_getResult
} component Judge
```

Decision tables nicely exploit the projectional editor.

Features

Combinable Extensions!

```
exported component Judge extends nothing {
  provides FlightJudger judger
  int16 points = 0;
  void judger_reset() <= op judger.reset {
    points = 0;
  } runnable judger_reset
  void judger_addTrackpoint(Trackpoint* tp) <= op judger.addTrackpoint {
    points += 0
```

	tp->alt <= 2000 m	tp->alt >= 2000 m
tp->speed < 150 mps	0	10
tp->speed >= 150 mps	5	20

```
  } runnable judger_addTrackpoint
  int16 judger_getResult() <= op judger.getResult {
    return points;
  } runnable judger_getResult
} component Judge
```

C, components, units and decision tables combined!

Features

Decision Tables II

```
SUCCESS: Table complete.
FAIL: cells (1, 1) and (2, 1) are inconsistent.
  tp.id : 0
  tp.timestamp : 0
  tp.x : 0
  tp.y : 0
  tp.speed : 0
  tp.alt : 2000
FAIL: cells (1, 2) and (2, 2) are inconsistent.
  tp.id : 0
  tp.timestamp : 0
  tp.x : 0
  tp.y : 0
  tp.speed : 150
  tp.alt : 2000
```

Decision Tables are analyzed f. consistency and completeness

Features

State Machines I

```
statemachine FlightAnalyzer initial = beforeFlight {
  state beforeFlight {  }
  state airborne {  }
  state landing {  }
  state landed {  }
  state crashed {  }
}
```

State machines fundamentally consist of states.

Features

State Machines II

```
state beforeFlight {
  entry { points = 0; }
  on next [tp->alt > 0 m] -> airborne
  exit { points += TAKEOFF; }
}
state airborne {
  on next [tp->alt == 0 m && tp->speed == 0 mps] -> crashed
  on next [tp->alt == 0 m && tp->speed > 0 mps] -> landing
  on next [tp->speed > 200 mps]
     -> airborne { points += VERY_HIGH_SPEED; }
  on next [tp->speed > 100 mps]
     -> airborne { points += HIGH_SPEED; }
  on reset [ ] -> beforeFlight
}
state landing {
  on next [tp->speed == 0 mps] -> landed
  on next [ ] -> landing { points--; }
  on reset [ ] -> beforeFlight
}
```

States contain transitions with guards, and actions.

Features

State Machines III

```
test case testFlightAnalyzer {
  FlightAnalyzer f;
  sminit(f);
}

test case testFlightAnalyzer {
  FlightAnalyzer f;
  sminit(f);
  assert(0) smIsInState(f, beforeFlight);
  smtrigger(f, next(makeTP(100, 100)));
  assert(3) smIsInState(f, airborne) && f.points == 100;
  ...
}
```

State machines can be instantiated; code can interact.

Features

State Machines IV

```
test case testFlightAnalyzer {
  FlightAnalyzer f;
  sminit(f);
}

test case testFlightAnalyzer {
  test statemachine f {
    next(makeTP(200, 100)) -> airborne
    next(makeTP(300, 150)) -> airborne
    next(makeTP(0, 90)) -> landing
    next(makeTP(0, 0)) -> landed
  }
}
```

+ special support for testing state machines.

Features

State Machines V

```
statemachine FlightAnalyzer initial = beforeFlight {
  ...
  state crashed {
    entry { raiseAlarm(); }
  }
}
...
void raiseAlarm() {}
```

```
statemachine FlightAnalyzer initial = beforeFlight {
  out crashNotification() => raiseAlarm
  ...
  state crashed {
    entry { send crashNotification(); }
  }
}
```

Outgoing interactins via function calls or out events.

Features

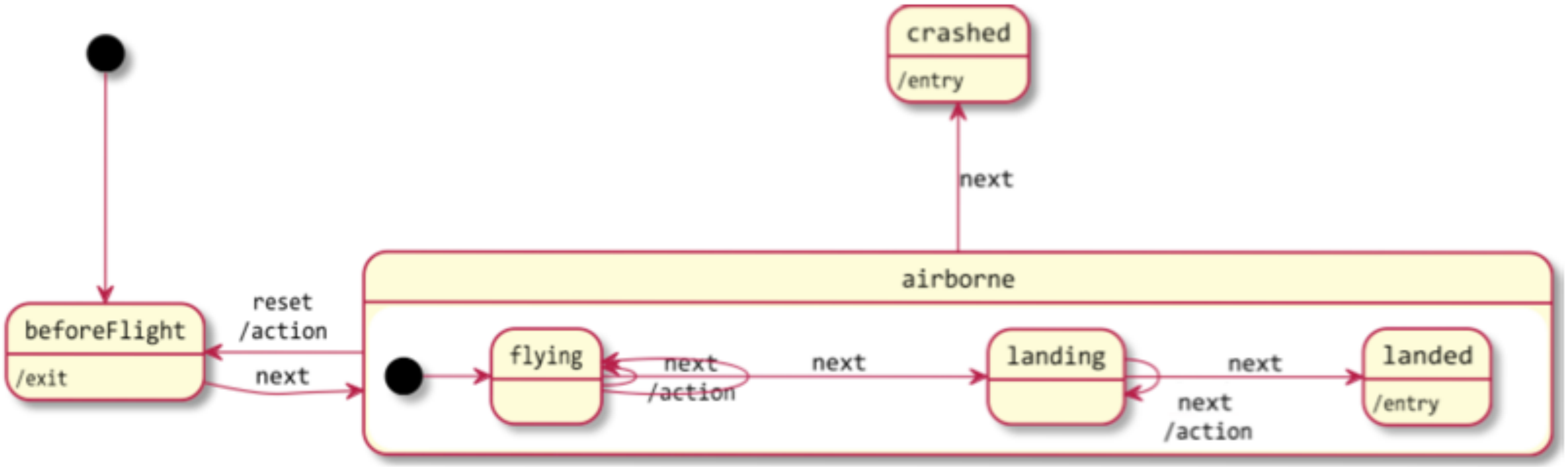
State Machines VI

```
composite state airborne initial = flying {
  on reset [ ] -> beforeFlight { points = 0; }
  on next [tp->alt == 0 m && tp->speed == 0 mps] -> crashed
  state flying {
    on next [tp->alt == 0 m && tp->speed > 0 mps] -> landing
    on next [tp->speed > 200 mps]
       -> flying { points += VERY_HIGH_SPEED; }
    on next [tp->speed > 100 mps]
       -> flying { points += HIGH_SPEED; }
  }
  state landing {
    on next [tp->speed == 0 mps] -> landed
    on next [ ] -> landing { points--; }
  }
  state landed {
    entry { points += LANDING; }
  }
```

Hierarchical state machines (composite states)

Features

State Machines VII

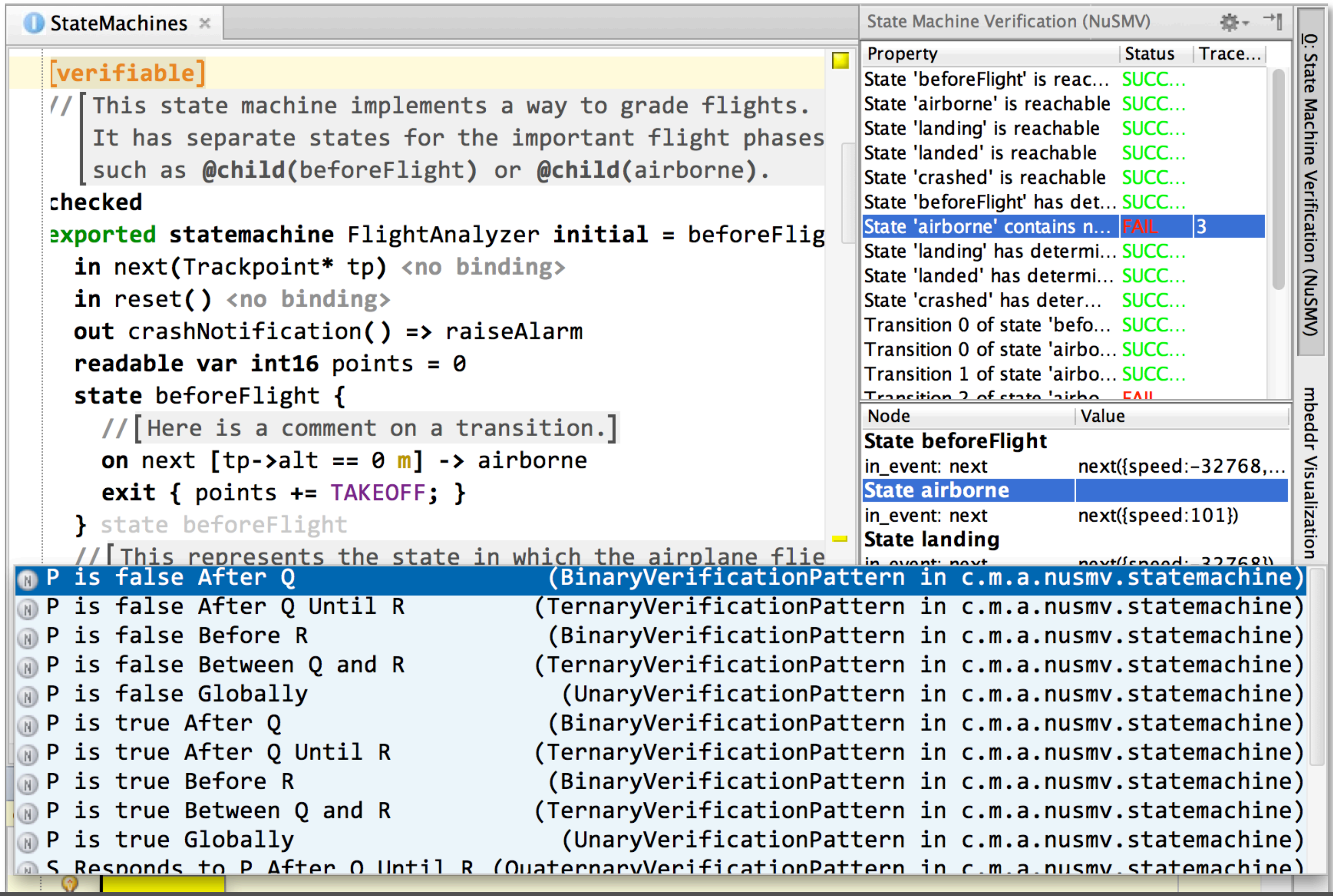


State Machines can be visualized in various way.

Features

mbeddr

State Machines VIII



StateMachines
State Machine Verification (NuSMV)
0: State Machine Verification (NuSMV)
mbeddr Visualization

```
[verifiable]
//[This state machine implements a way to grade flights.
  It has separate states for the important flight phases
  such as @child(beforeFlight) or @child(airborne).
checked
exported statemachine FlightAnalyzer initial = beforeFlig
  in next(Trackpoint* tp) <no binding>
  in reset() <no binding>
  out crashNotification() => raiseAlarm
  readable var int16 points = 0
  state beforeFlight {
    //[Here is a comment on a transition.]
    on next [tp->alt == 0 m] -> airborne
    exit { points += TAKEOFF; }
  } state beforeFlight
  //[This represents the state in which the airplane flie
```

Property	Status	Trace...
State 'beforeFlight' is reac...	SUCC...	
State 'airborne' is reachable	SUCC...	
State 'landing' is reachable	SUCC...	
State 'landed' is reachable	SUCC...	
State 'crashed' is reachable	SUCC...	
State 'beforeFlight' has det...	SUCC...	
State 'airborne' contains n...	FAIL	3
State 'landing' has determi...	SUCC...	
State 'landed' has determi...	SUCC...	
State 'crashed' has deter...	SUCC...	
Transition 0 of state 'befo...	SUCC...	
Transition 0 of state 'airbo...	SUCC...	
Transition 1 of state 'airbo...	SUCC...	
Transition 2 of state 'airbo...	FAIL	

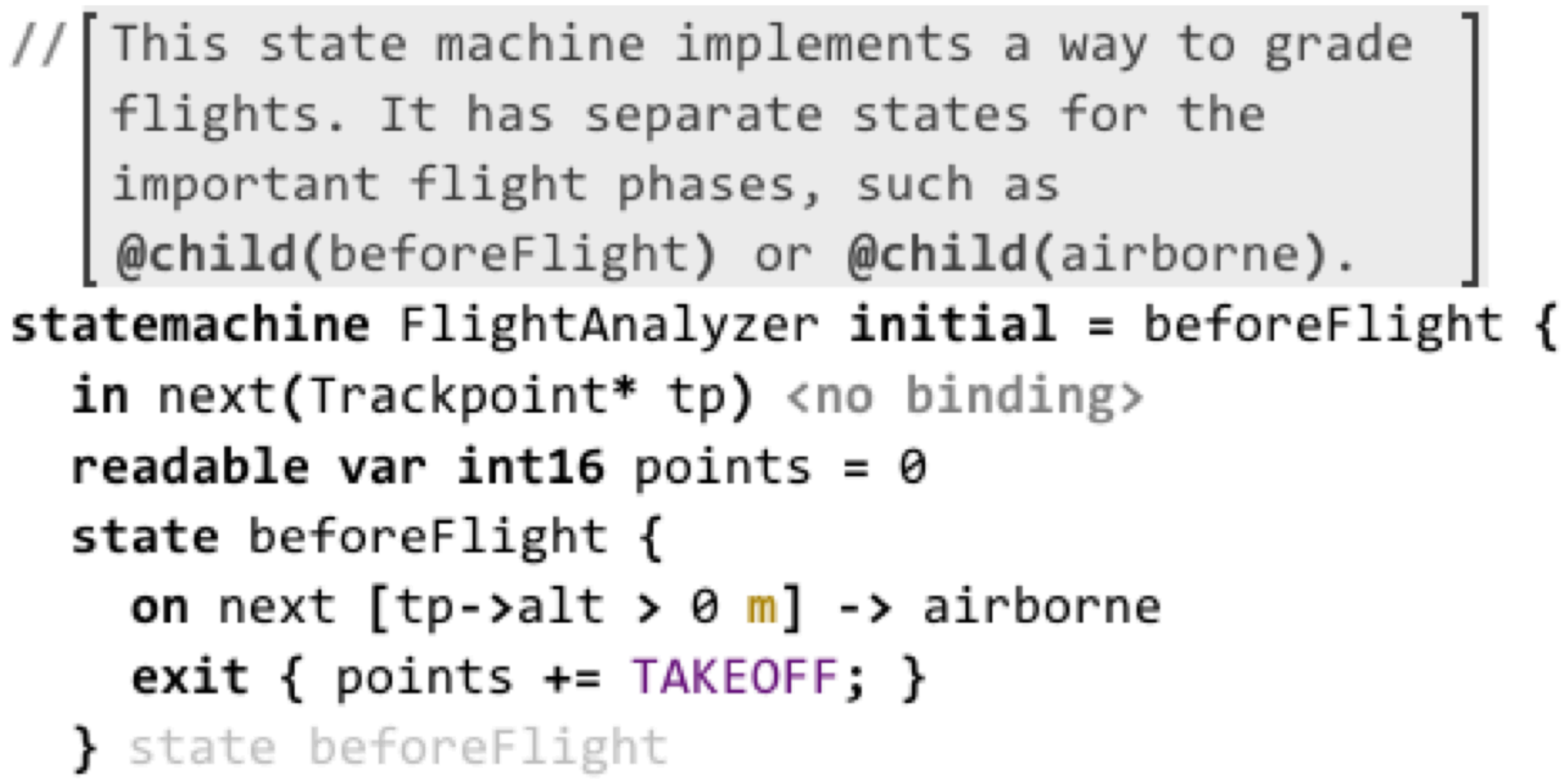
Node	Value
State beforeFlight	
in_event: next	next({speed:-32768,...
State airborne	
in_event: next	next({speed:101})
State landing	
in_event: next	next({speed:-32768})

```
P is false After Q               (BinaryVerificationPattern in c.m.a.nusmv.statemachine)
P is false After Q Until R      (TernaryVerificationPattern in c.m.a.nusmv.statemachine)
P is false Before R              (BinaryVerificationPattern in c.m.a.nusmv.statemachine)
P is false Between Q and R      (TernaryVerificationPattern in c.m.a.nusmv.statemachine)
P is false Globally                (UnaryVerificationPattern in c.m.a.nusmv.statemachine)
P is true After Q                (BinaryVerificationPattern in c.m.a.nusmv.statemachine)
P is true After Q Until R       (TernaryVerificationPattern in c.m.a.nusmv.statemachine)
P is true Before R               (BinaryVerificationPattern in c.m.a.nusmv.statemachine)
P is true Between Q and R       (TernaryVerificationPattern in c.m.a.nusmv.statemachine)
P is true Globally                 (UnaryVerificationPattern in c.m.a.nusmv.statemachine)
S Responds to P After Q Until R (QuaternaryVerificationPattern in c.m.a.nusmv.statemachine)
```

Symbolic Model Checking for State Machines

Features

Documentation



```
//[This state machine implements a way to grade  ]
  [flights. It has separate states for the       ]
  [important flight phases, such as              ]
  [@child(beforeFlight) or @child(airborne).     ]
statemachine FlightAnalyzer initial = beforeFlight {
  in next(Trackpoint* tp) <no binding>
  readable var int16 points = 0
  state beforeFlight {
    on next [tp->alt > 0 m] -> airborne
    exit { points += TAKEOFF; }
  } state beforeFlight
```

Rich, Structured Comments (note the embedded nodes)

Features

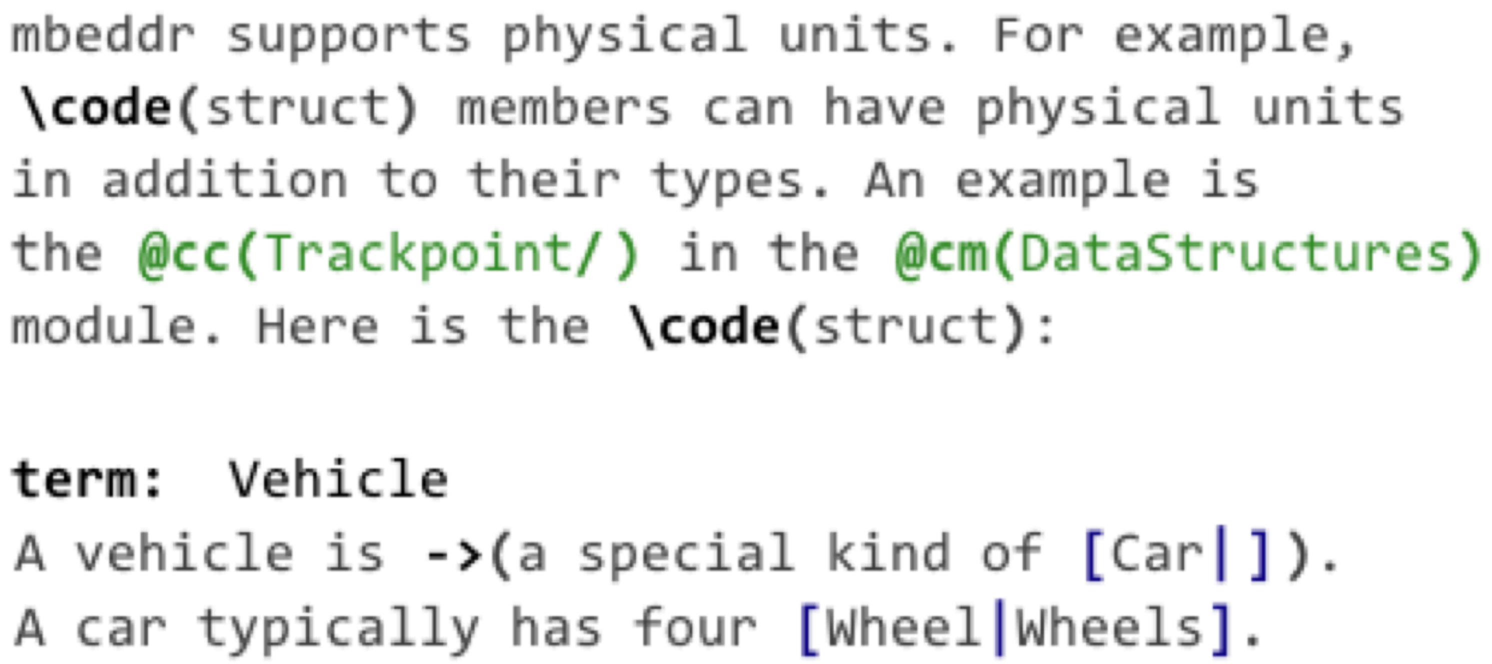
Documentation II

```
section 1.2 existing.comps: Interfaces and Components {

  [ Interfaces declare operations that can be provided or used by components.
    Each operation can also declare pre- and postconditions as well as
    protocols. These can be checked either at runtime or statically. The
    @cm(Components) module contains examples. Below is an interface: ]

  embed as text Components.TrackpointStore1/

  [ The interfaces, components and thei relationships in a given module can
    also be rendered graphically. An example is shown in @fig(ci) ]

  visualize Components.store.TrackpointStore1/
      components + interfaces (grouped) as ci
    location: vis:/
    scaling:  width100
  [The components and their provided (solid lines) and required (dotted lines)
   ports. ]

  [ Of course the visualizations are also not just images. In the source to the
    document, we embed references to \code(IVisualizable) instances. In the
    doc, one can select the visualization category, and then, during
    generation, PlantUML automatically rerenders the image. ]

}
```

Documentation Language w/ Embeddable Code (LaTeX/HTML)

Features

Documentation III

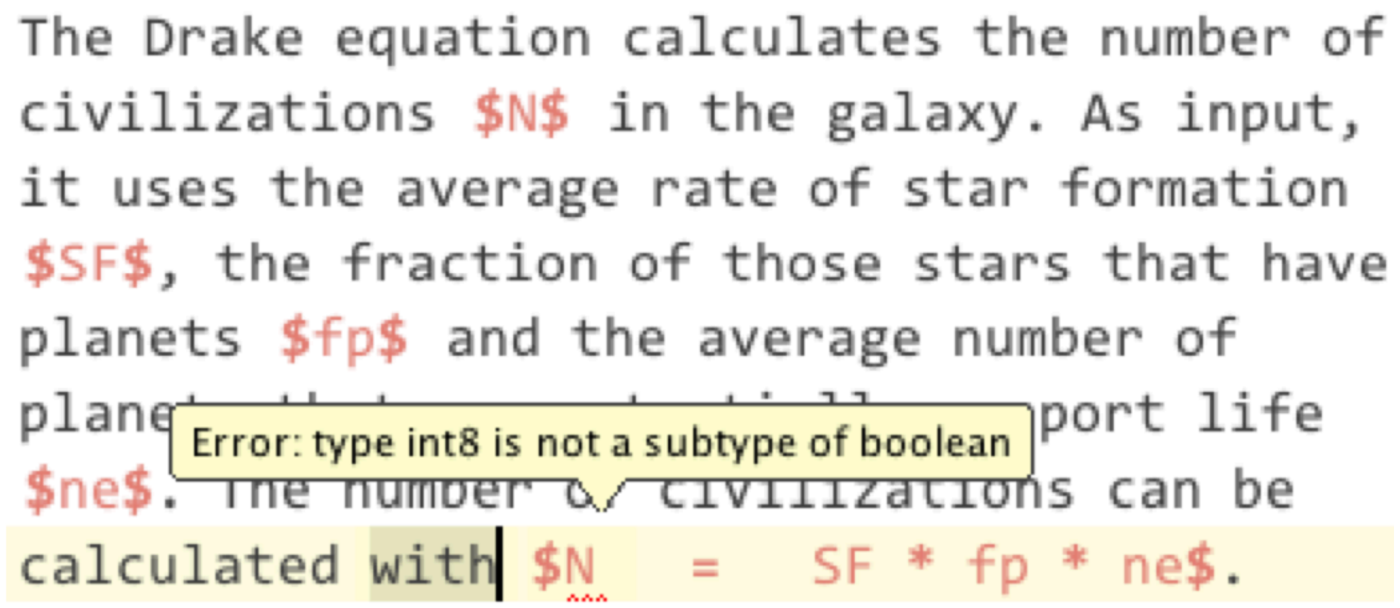


```
mbeddr supports physical units. For example,
\code(struct) members can have physical units
in addition to their types. An example is
the @cc(Trackpoint/) in the @cm(DataStructures)
module. Here is the \code(struct):

term:  Vehicle
A vehicle is ->(a special kind of [Car|]).
A car typically has four [Wheel|Wheels].
```

Format Text, Reference Code, Define Glossary Terms

Features

Documentation IV



The Drake equation calculates the number of civilizations N in the galaxy. As input, it uses the average rate of star formation SF, the fraction of those stars that have planets fp and the average number of plane[illegible]port life ne. The number o[illegible] civilizations can be calculated with $N = SF * fp * ne$.

Error: type int8 is not a subtype of boolean

Features

Product Line Variability

```
feature model FlightProcessor
  processing ? {
    nullify
    normalizeSpeed xor {
      maxCustom [int16/mps/ maxSpeed]
      max100
    }
  }
configuration model cfgNullifyMaxAt200 configures FlightProcessor
  processing {
    nullify
    normalizeSpeed {
      maxCustom [maxSpeed = 200 mps]
    }
  }
```

Feature Models (and Checked Configs) to Express Variability

Features

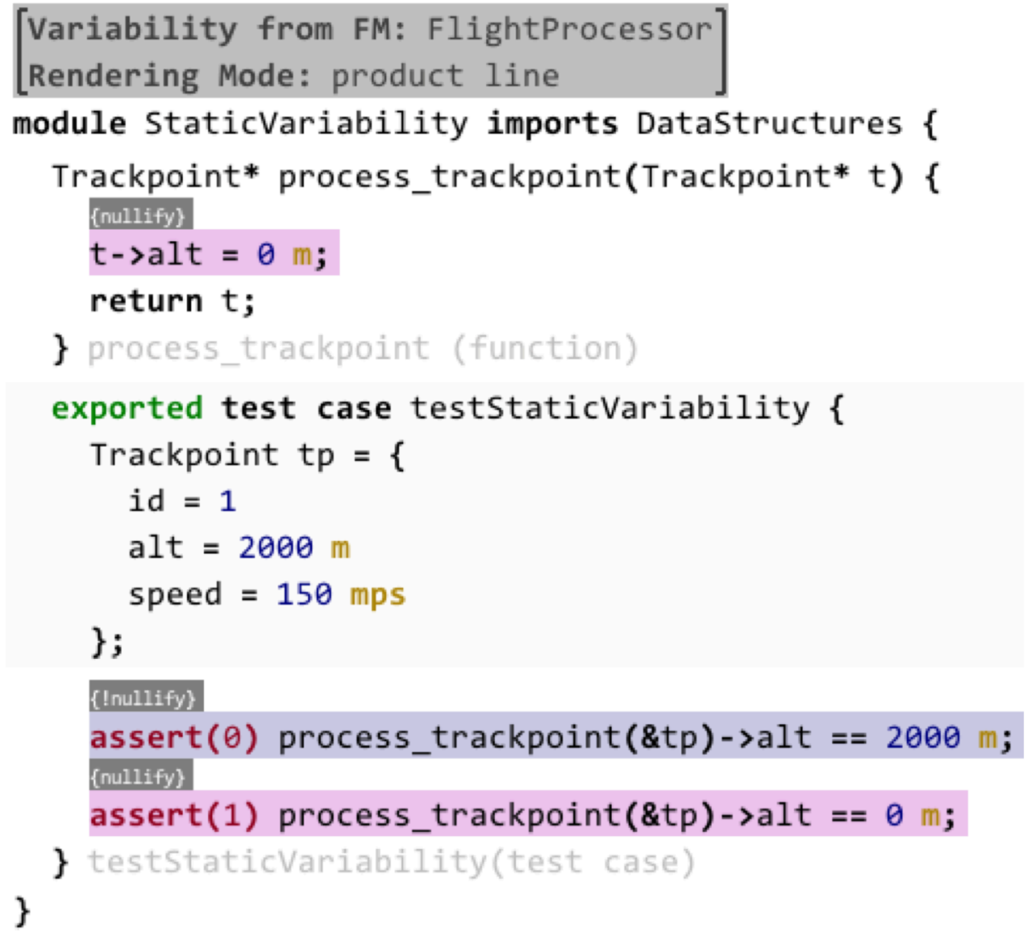
Product Line Variability II

```
Trackpoint processTrackpoint(fmconfig<FlightProcessor> cfg,
                             Trackpoint tp) {
  Trackpoint result;
  variant<cfg> {
    case (nullify && maxCustom) {
      result = process_nullifyAlt(tp);
      if (tp.speed > maxCustom.maxSpeed) {
        result.speed = maxCustom.maxSpeed;
      }
    }
    case (nullify && max100) {
    }
    case (nullify) { result = process_nullifyAlt(tp); }
    default { result = process_doNothing(tp); }
  }
  return result;
```

Runtime Variability based on Feature Models

Features

Product Line Variability III

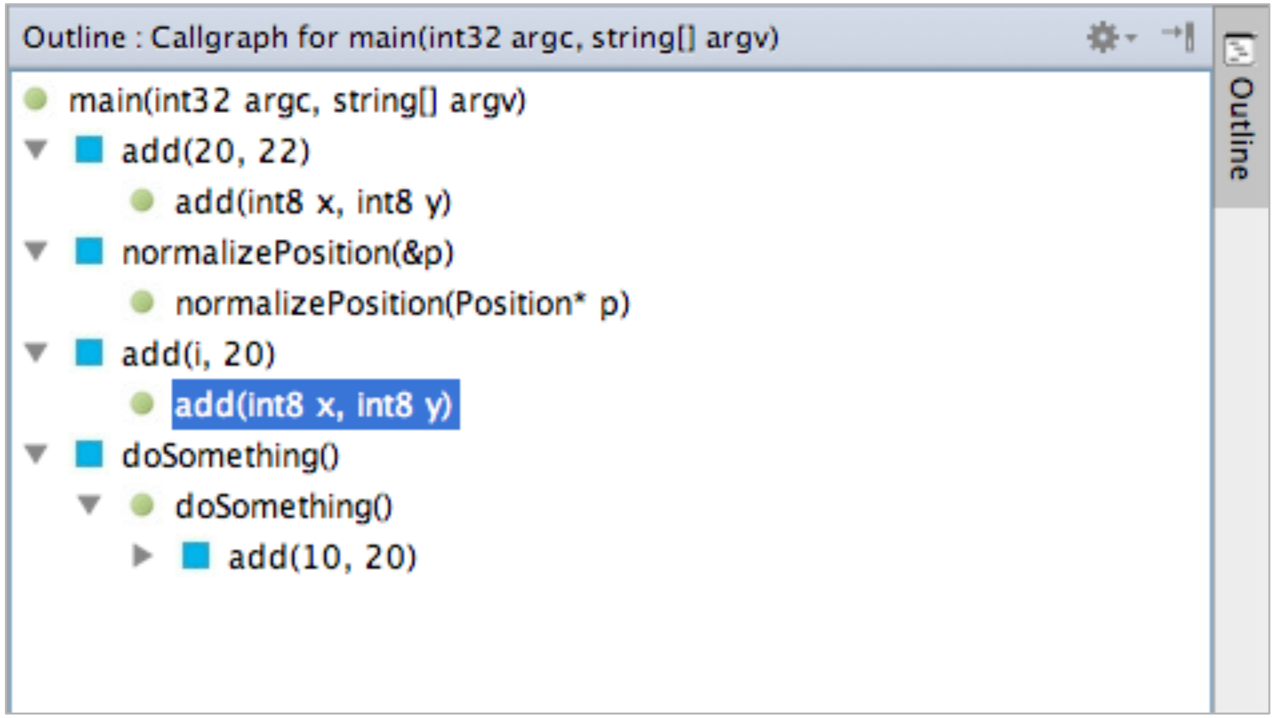


```
[Variability from FM: FlightProcessor]
[Rendering Mode: product line        ]
module StaticVariability imports DataStructures {
  Trackpoint* process_trackpoint(Trackpoint* t) {
    {nullify}
    t->alt = 0 m;
    return t;
  } process_trackpoint (function)
  exported test case testStaticVariability {
    Trackpoint tp = {
      id = 1
      alt = 2000 m
      speed = 150 mps
    };
    {!nullify}
    assert(0) process_trackpoint(&tp)->alt == 2000 m;
    {nullify}
    assert(1) process_trackpoint(&tp)->alt == 0 m;
  } testStaticVariability(test case)
}
```

Static Variability for any Program w/ Variant Editing

Features

Tree Views I



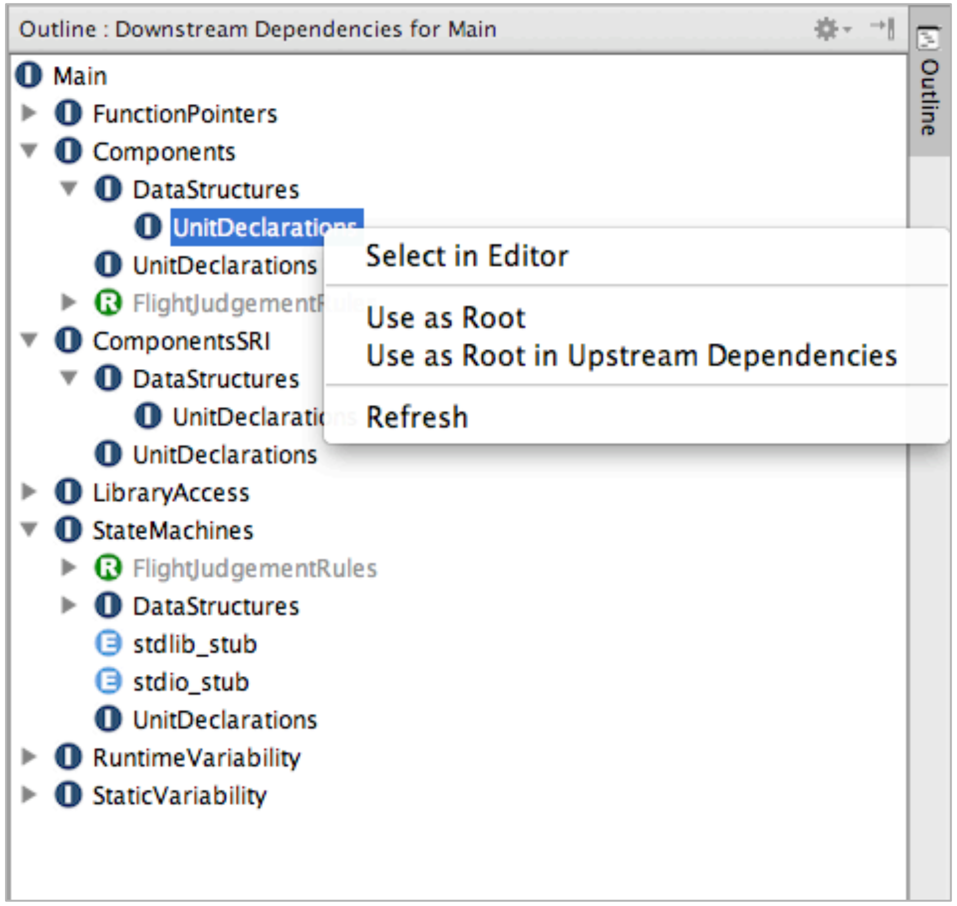
Outline : Callgraph for main(int32 argc, string[] argv)

```
● main(int32 argc, string[] argv)
▼ ■ add(20, 22)
      ● add(int8 x, int8 y)
▼ ■ normalizePosition(&p)
      ● normalizePosition(Position* p)
▼ ■ add(i, 20)
      ● add(int8 x, int8 y)
▼ ■ doSomething()
   ▼ ● doSomething()
      ▶ ■ add(10, 20)
```

Different Tree View Structures defined by language concepts.

Features

Tree Views II


Outline : Downstream Dependencies for Main

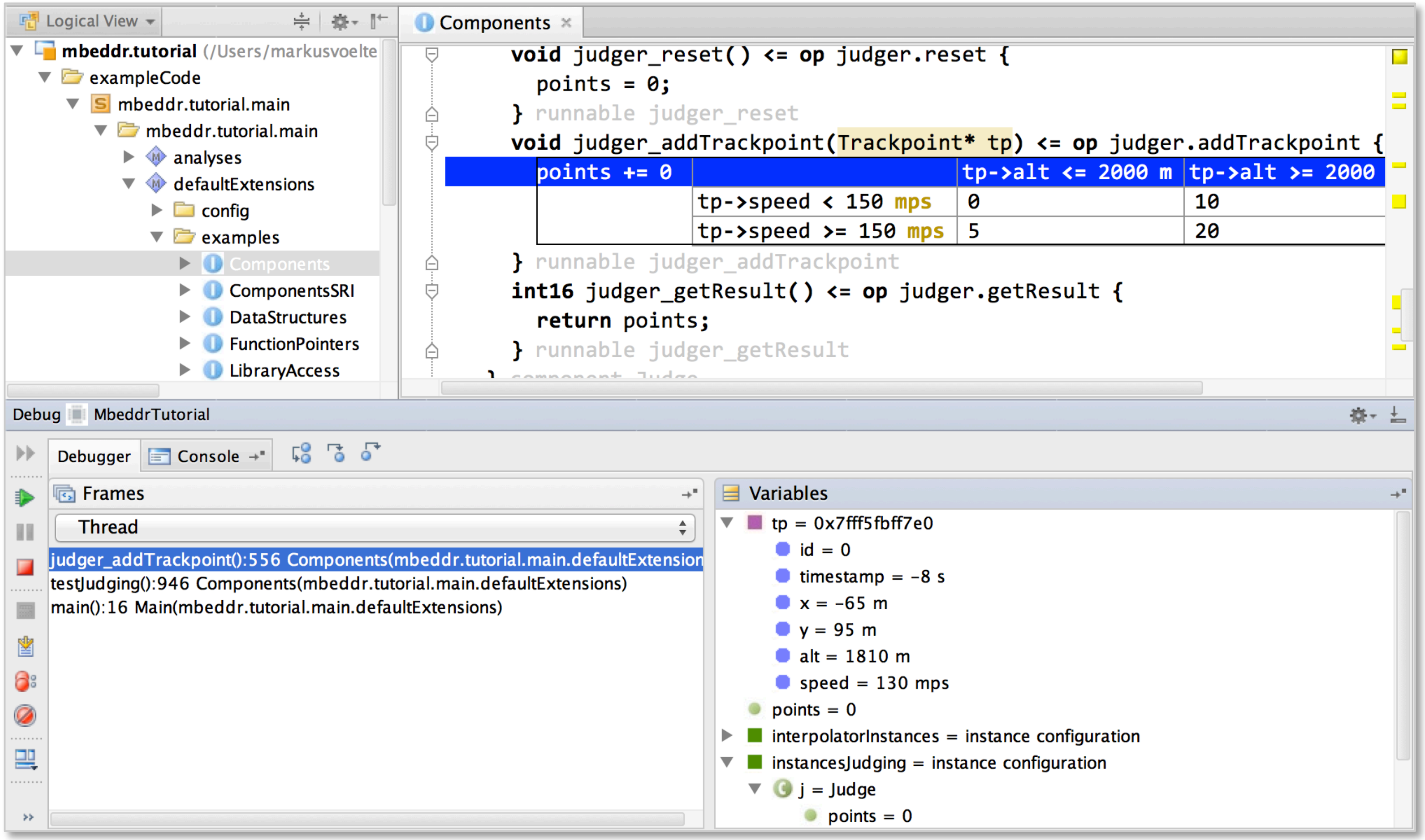
Outline

Main
FunctionPointers
Components
DataStructures
UnitDeclarations
UnitDeclarations
FlightJudgementRules
ComponentsSRI
DataStructures
UnitDeclarations
UnitDeclarations
LibraryAccess
StateMachines
FlightJudgementRules
DataStructures
stdlib_stub
stdio_stub
UnitDeclarations
RuntimeVariability
StaticVariability

Custom Commands are supported as well.

Features

Debugging



Logical View

- mbeddr.tutorial (/Users/markusvoelte
  - exampleCode
    - mbeddr.tutorial.main
      - mbeddr.tutorial.main
        - analyses
        - defaultExtensions
          - config
          - examples
            - Components
            - ComponentsSRI
            - DataStructures
            - FunctionPointers
            - LibraryAccess

Components

```
    void judger_reset() <= op judger.reset {
      points = 0;
    } runnable judger_reset
    void judger_addTrackpoint(Trackpoint* tp) <= op judger.addTrackpoint {
```

points += 0		tp->alt <= 2000 m	tp->alt >= 2000
	tp->speed < 150 mps	0	10
	tp->speed >= 150 mps	5	20

```
    } runnable judger_addTrackpoint
    int16 judger_getResult() <= op judger.getResult {
      return points;
    } runnable judger_getResult
```

Debug MbeddrTutorial

Debugger Console

Frames

Thread

```
judger_addTrackpoint():556 Components(mbeddr.tutorial.main.defaultExtension
testJudging():946 Components(mbeddr.tutorial.main.defaultExtensions)
main():16 Main(mbeddr.tutorial.main.defaultExtensions)
```

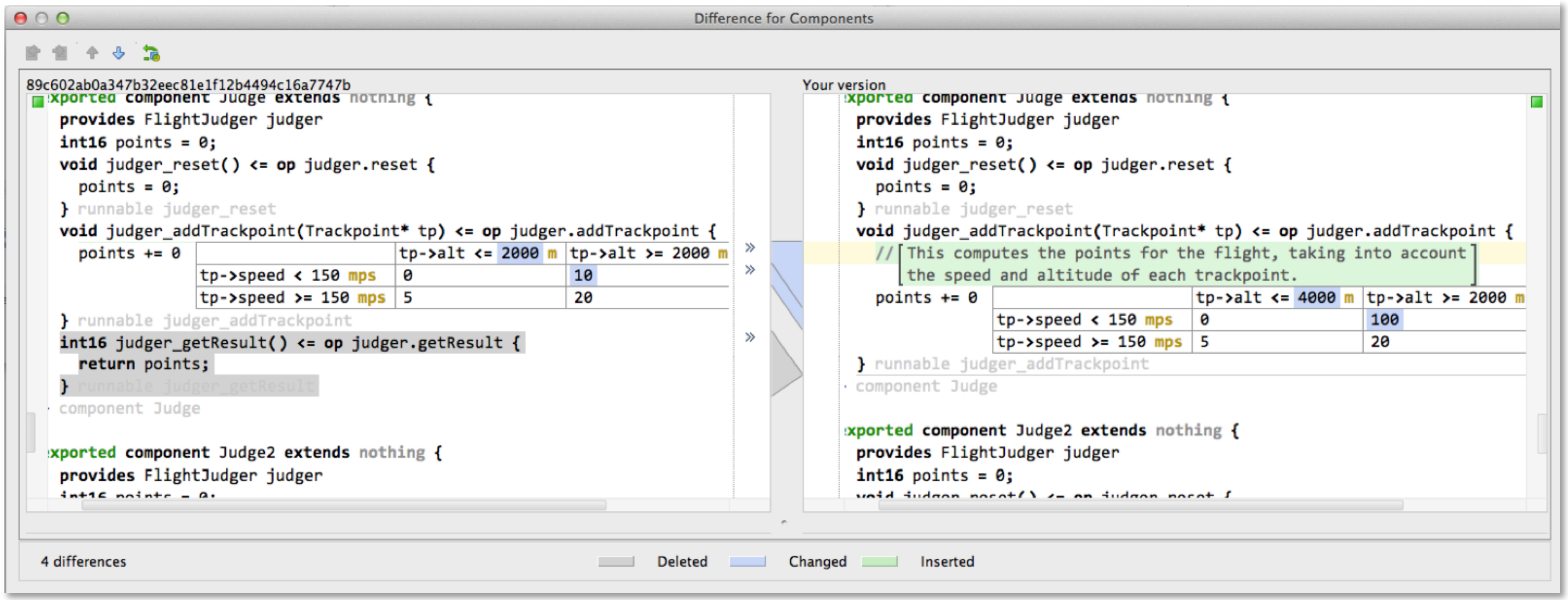
Variables

```
tp = 0x7fff5fbff7e0
    id = 0
    timestamp = -8 s
    x = -65 m
    y = 95 m
    alt = 1810 m
    speed = 130 mps
points = 0
interpolatorInstances = instance configuration
instancesJudging = instance configuration
    j = Judge
        points = 0
```

Debugging on the DSL Level (Extensible!)

Features

VCS Diff/Merge



Difference for Components

89c602ab0a347b32eec81e1f12b4494c16a7747b

```
xported component Judge extends nothing {
  provides FlightJudger judger
  int16 points = 0;
  void judger_reset() <= op judger.reset {
    points = 0;
  } runnable judger_reset
  void judger_addTrackpoint(Trackpoint* tp) <= op judger.addTrackpoint {
    points += 0 |                    | tp->alt <= 2000 m | tp->alt >= 2000 m
                | tp->speed < 150 mps  | 0               | 10
                | tp->speed >= 150 mps | 5               | 20
  } runnable judger_addTrackpoint
  int16 judger_getResult() <= op judger.getResult {
    return points;
  } runnable judger_getResult
component Judge

xported component Judge2 extends nothing {
  provides FlightJudger judger
```

Your version

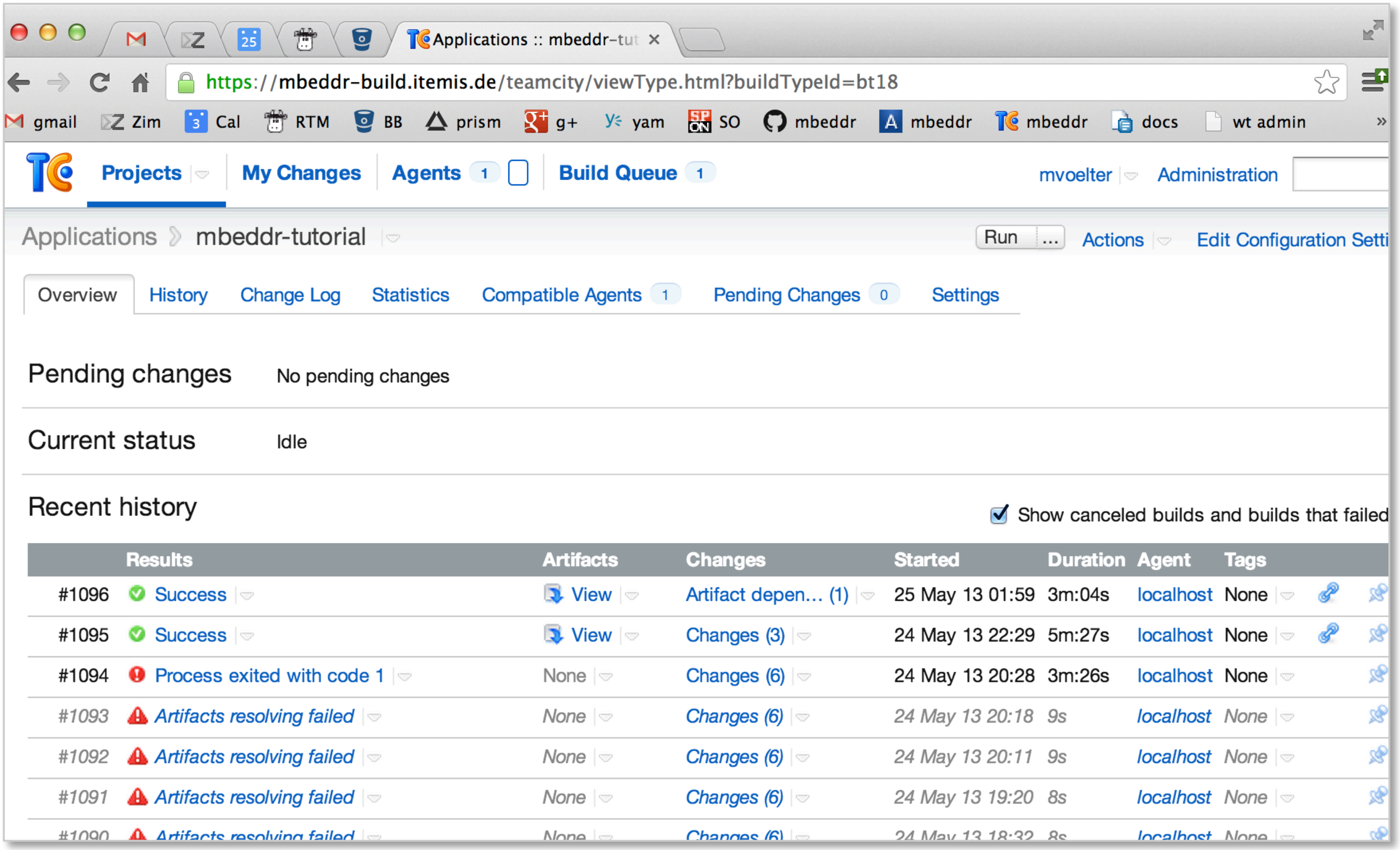
```
xported component Judge extends nothing {
  provides FlightJudger judger
  int16 points = 0;
  void judger_reset() <= op judger.reset {
    points = 0;
  } runnable judger_reset
  void judger_addTrackpoint(Trackpoint* tp) <= op judger.addTrackpoint {
    // This computes the points for the flight, taking into account
       the speed and altitude of each trackpoint.
    points += 0 |                    | tp->alt <= 4000 m | tp->alt >= 2000 m
                | tp->speed < 150 mps  | 0               | 100
                | tp->speed >= 150 mps | 5               | 20
  } runnable judger_addTrackpoint
component Judge

xported component Judge2 extends nothing {
  provides FlightJudger judger
  int16 points = 0;
```

4 differences   ▭ Deleted   ▭ Changed   ▭ Inserted

Diff/Merge on the Projected Syntax

Features

CI Server Integration



https://mbeddr-build.itemis.de/teamcity/viewType.html?buildTypeId=bt18

Applications › mbeddr-tutorial

Overview | History | Change Log | Statistics | Compatible Agents 1 | Pending Changes 0 | Settings

Pending changes — No pending changes

Current status — Idle

Recent history — ✓ Show canceled builds and builds that failed

	Results	Artifacts	Changes	Started	Duration	Agent	Tags
#1096	Success	View	Artifact depen... (1)	25 May 13 01:59	3m:04s	localhost	None
#1095	Success	View	Changes (3)	24 May 13 22:29	5m:27s	localhost	None
#1094	Process exited with code 1	None	Changes (6)	24 May 13 20:28	3m:26s	localhost	None
#1093	*Artifacts resolving failed*	*None*	*Changes (6)*	*24 May 13 20:18*	*9s*	*localhost*	*None*
#1092	*Artifacts resolving failed*	*None*	*Changes (6)*	*24 May 13 20:11*	*9s*	*localhost*	*None*
#1091	*Artifacts resolving failed*	*None*	*Changes (6)*	*24 May 13 19:20*	*8s*	*localhost*	*None*
#1090	*Artifacts resolving failed*	*None*	*Changes (6)*	*24 May 13 18:32*	*8s*	*localhost*	*None*

Building Programs on Command Line/CI Server

3

Generic Tools **GTSL** Specific Languages

Thought Process

From Data Formats To Languages

$$\frac{\dfrac{\textit{Structure, Constraints, Semantics}}{\textbf{Data Format} + \textit{Syntax} + \textit{IDE}}}{\textbf{Language}}$$

Thought Process

Language Engineering

Languages

Language Reuse
Language Modularization
Language Composition

Language Engineering

Thought Process

Language Engineering

Languages

Language Engineering

Text	*Math*	*Graphics*
Tables	*Symbols*	*Forms*

Syntactic Diversity

Thought Process

Language Workbenches

Languages

Language Engineering

Syntactic Diversity

But does this really work?

Language Workbenches

Generic Tools, Specific Languages

Ingredients

Specific Languages

Languages

Language Engineering

Syntactic Diversity

Generic Tools

Language Workbenches

Generic Tools, Specific Languages

Ingredients

Specific Languages

Languages

Language Engineering

Syntactic Diversity

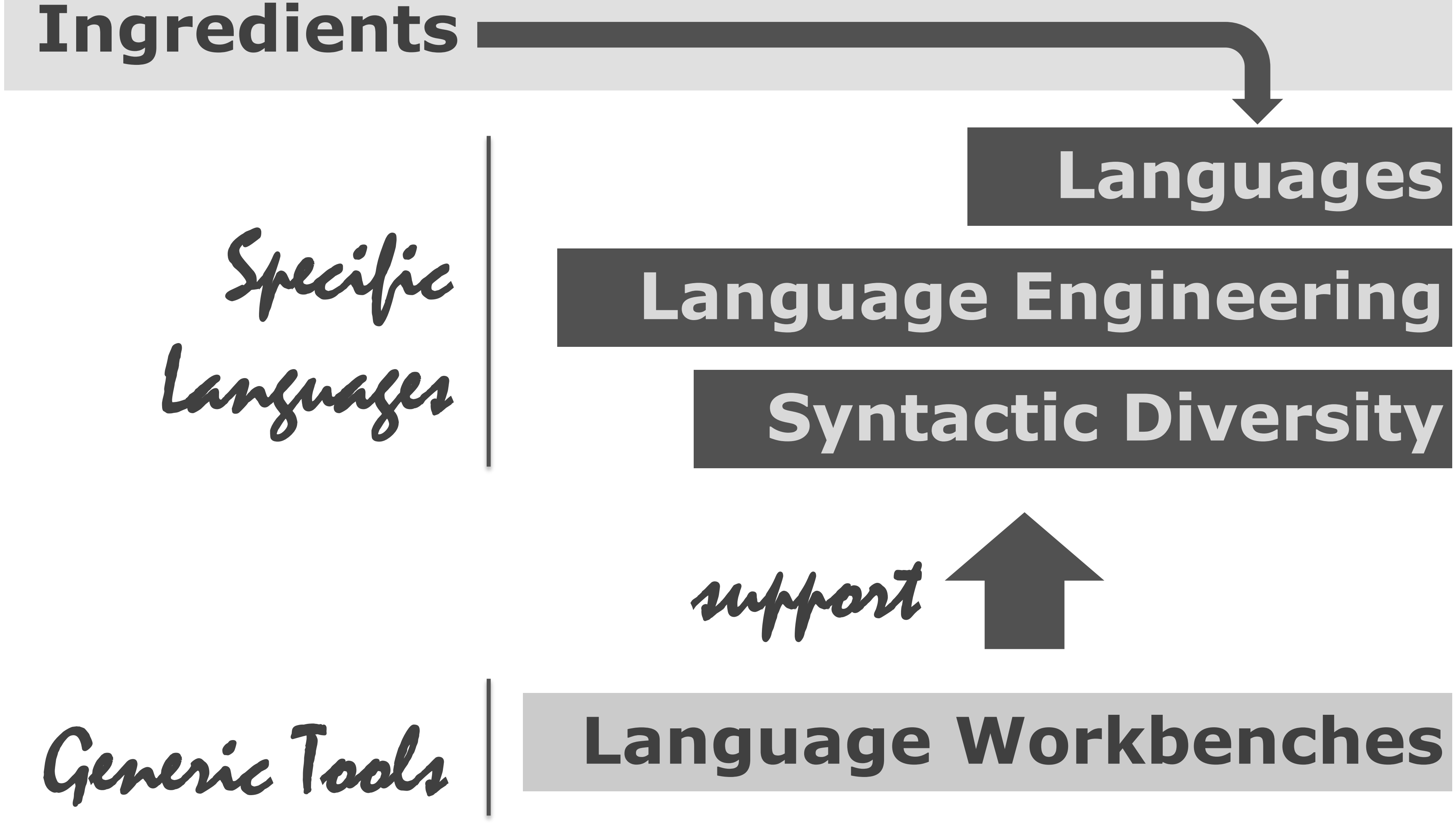
Generic Tools

Language Workbenches

(we don't have to reimplement editors and synchronizers)

Generic Tools, Specific Languages

Ingredients



Language Workbenches

Typical Features

Goto Definition/Find Usages

Error Markup/Quick Fixes

Syntax Highlighting

Code Completion

Search/Replace

Refactoring

Debugging

Reporting

Visualization

Version Control

Language Workbenches

Typical Features

for any Language!

Language Workbenches

Typical Features

Language Workbenches act as the foundation for IDEs for any language.

Tool Extensibility

Study Findings I

The majority of our interviewees were very successful with MDE but all of them either **built their own** modeling tools, **made heavy adaptations** of off-the-shelf tools, or spent a lot of time finding ways to work around tools. The only accounts of easy-to-use, intuitive tools came from those who had developed tools themselves for bespoke purposes. Indeed, this suggests that current tools are a barrier to success rather than an enabler.

Jon Whittle, John Hutchinson, Mark Rouncefield, Hakan Burden, and Rogardt Heldal. Industrial Adoption of Model-Driven Engineering: Are the Tools Really the Problem? In *Proceedings of the 16th International Conference on Model Driven Engineering Languages and Systems (MODELS) 2013*. ACM, 2013.

Tool Extensibility

Study Findings II

Complexity problems are typically associated with off-the- shelf tools. Of particular note is **accidental complexity** – which can be introduced due to poor consideration of other categories, such as **lack of flexibility to adapt the tools** to a company's own context [..]

Jon Whittle, John Hutchinson, Mark Rouncefield, Hakan Burden, and Rogardt Heldal. Industrial Adoption of Model-Driven Engineering: Are the Tools Really the Problem? In *Proceedings of the 16th International Conference on Model Driven Engineering Languages and Systems (MODELS) 2013*. ACM, 2013.

Language Workbenches

Typical Features

Used by the tool vendor to
build the initial tool (languages).

Used by the end user to
adapt the tool (lang extensions)!

Extensions are first-class!

Generic Tools, Specific Languages

Adaptability is built-in!

Extensions are
first-class!

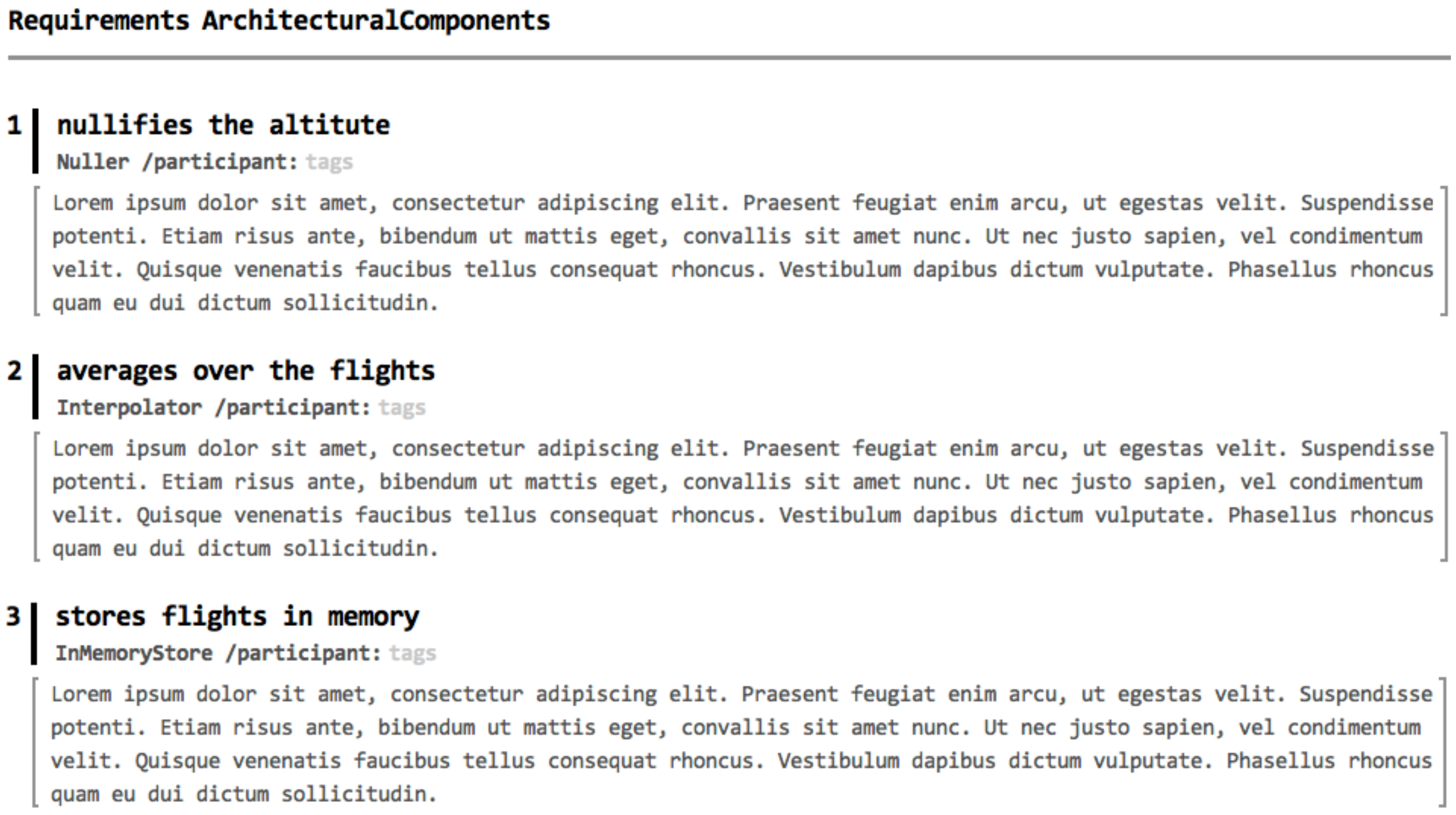
Fundamentally different from Today`s State-of-the-Art in Tools

4

Example II: Requirements

Features

Requirements



Requirements ArchitecturalComponents

1 **nullifies the altitute**
Nuller /participant: tags
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Praesent feugiat enim arcu, ut egestas velit. Suspendisse potenti. Etiam risus ante, bibendum ut mattis eget, convallis sit amet nunc. Ut nec justo sapien, vel condimentum velit. Quisque venenatis faucibus tellus consequat rhoncus. Vestibulum dapibus dictum vulputate. Phasellus rhoncus quam eu dui dictum sollicitudin.

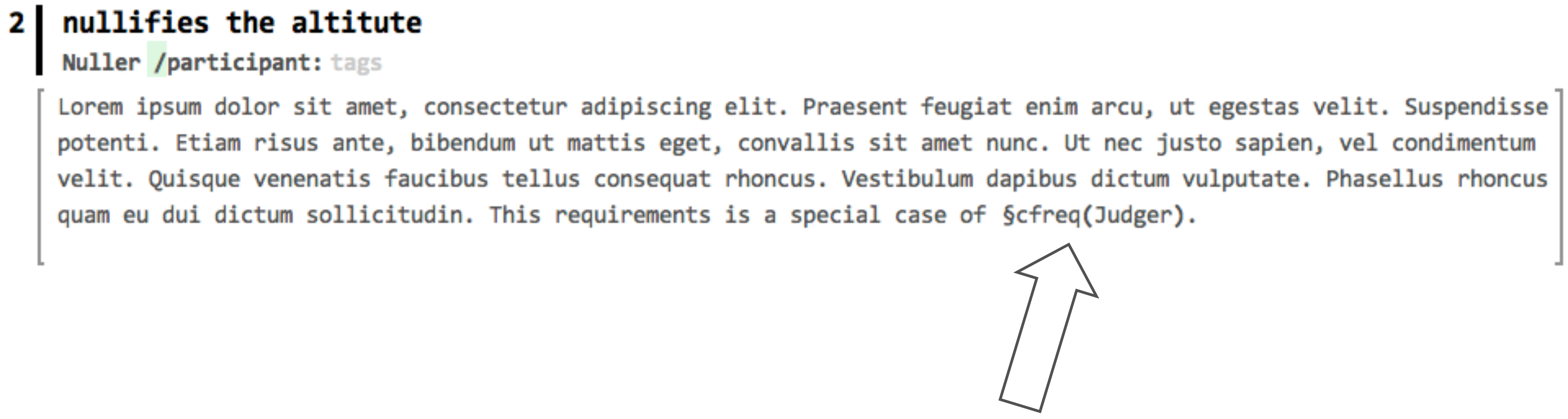
2 **averages over the flights**
Interpolator /participant: tags
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Praesent feugiat enim arcu, ut egestas velit. Suspendisse potenti. Etiam risus ante, bibendum ut mattis eget, convallis sit amet nunc. Ut nec justo sapien, vel condimentum velit. Quisque venenatis faucibus tellus consequat rhoncus. Vestibulum dapibus dictum vulputate. Phasellus rhoncus quam eu dui dictum sollicitudin.

3 **stores flights in memory**
InMemoryStore /participant: tags
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Praesent feugiat enim arcu, ut egestas velit. Suspendisse potenti. Etiam risus ante, bibendum ut mattis eget, convallis sit amet nunc. Ut nec justo sapien, vel condimentum velit. Quisque venenatis faucibus tellus consequat rhoncus. Vestibulum dapibus dictum vulputate. Phasellus rhoncus quam eu dui dictum sollicitudin.

Structured and Hierarchical Requirements.

Features

Requirements Relationships



2 **nullifies the altitute**

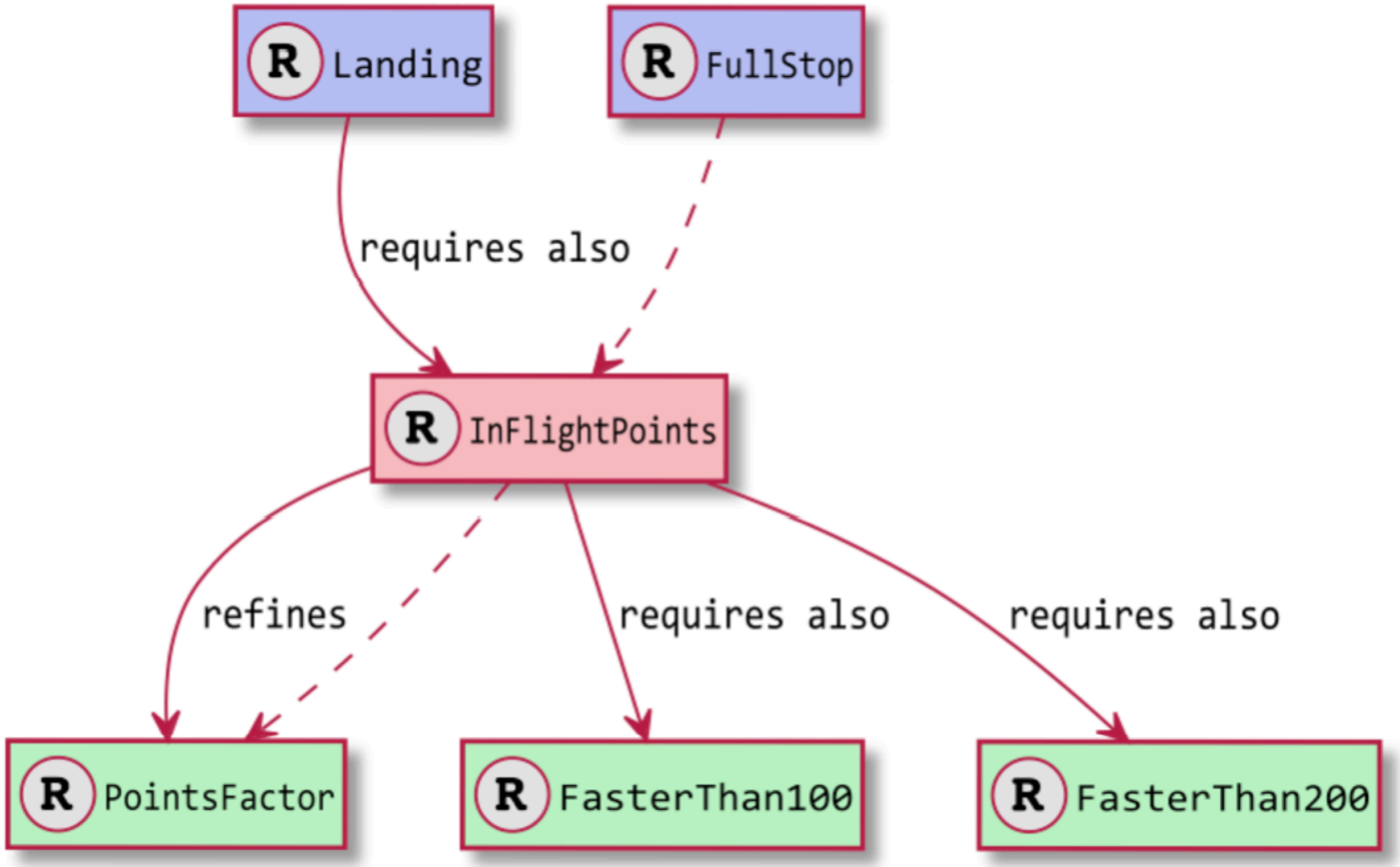
Nuller /participant: tags

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Praesent feugiat enim arcu, ut egestas velit. Suspendisse potenti. Etiam risus ante, bibendum ut mattis eget, convallis sit amet nunc. Ut nec justo sapien, vel condimentum velit. Quisque venenatis faucibus tellus consequat rhoncus. Vestibulum dapibus dictum vulputate. Phasellus rhoncus quam eu dui dictum sollicitudin. This requirements is a special case of §cfreq(Judger).

References to other Requirements (see Documentation lang.)

Features

Requirements Relationship Diagram

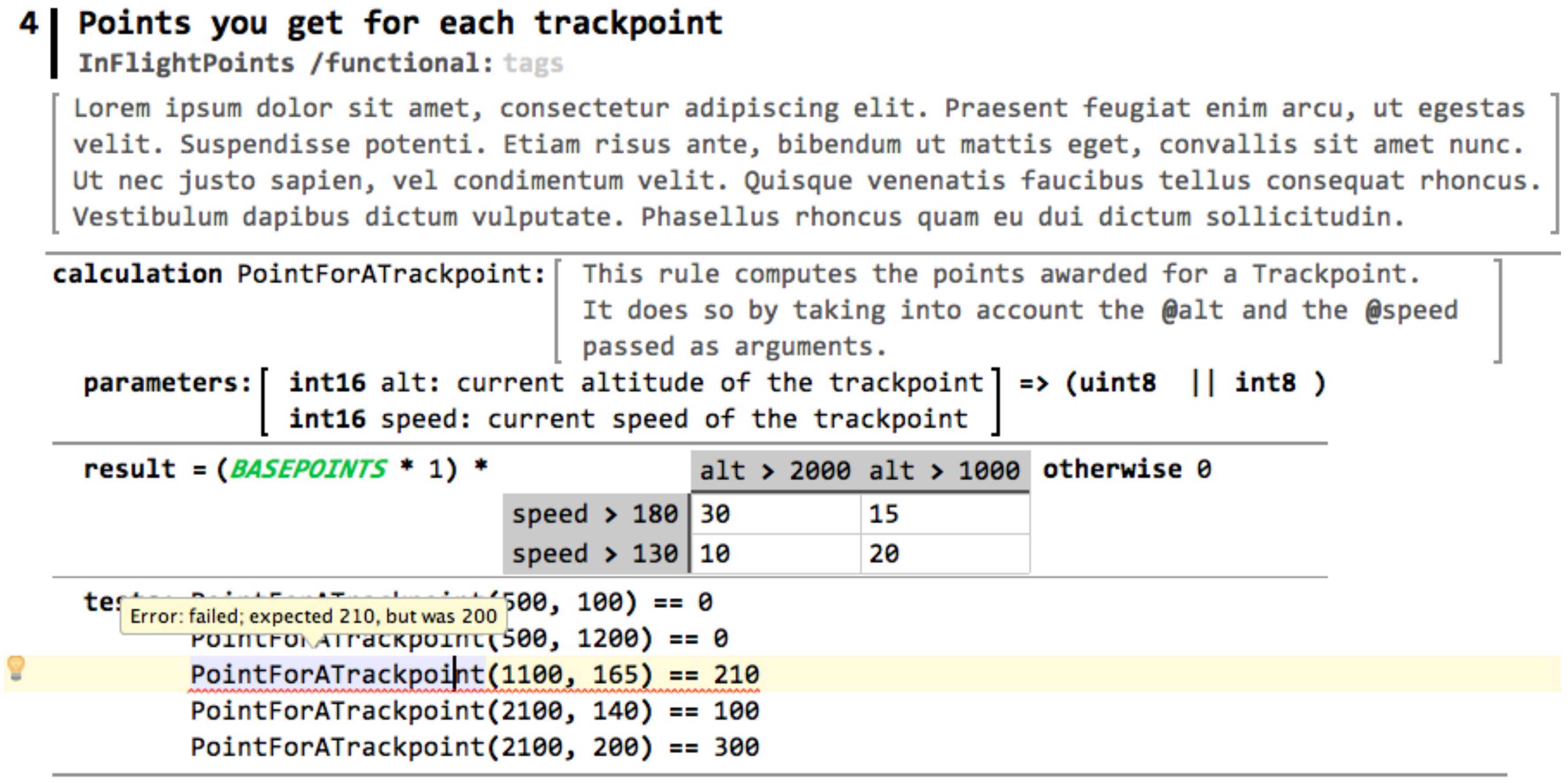


Relationships between Requirements (downstream, upstream)

Features

mbeddr

Requirements ext'd with Business Rules



4 **Points you get for each trackpoint**
InFlightPoints /functional: tags

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Praesent feugiat enim arcu, ut egestas velit. Suspendisse potenti. Etiam risus ante, bibendum ut mattis eget, convallis sit amet nunc. Ut nec justo sapien, vel condimentum velit. Quisque venenatis faucibus tellus consequat rhoncus. Vestibulum dapibus dictum vulputate. Phasellus rhoncus quam eu dui dictum sollicitudin.

```
calculation PointForATrackpoint: [ This rule computes the points awarded for a Trackpoint.
                                   It does so by taking into account the @alt and the @speed
                                   passed as arguments. ]
  parameters: [ int16 alt: current altitude of the trackpoint ] => (uint8  || int8 )
              [ int16 speed: current speed of the trackpoint  ]
  result = (BASEPOINTS * 1) *
```

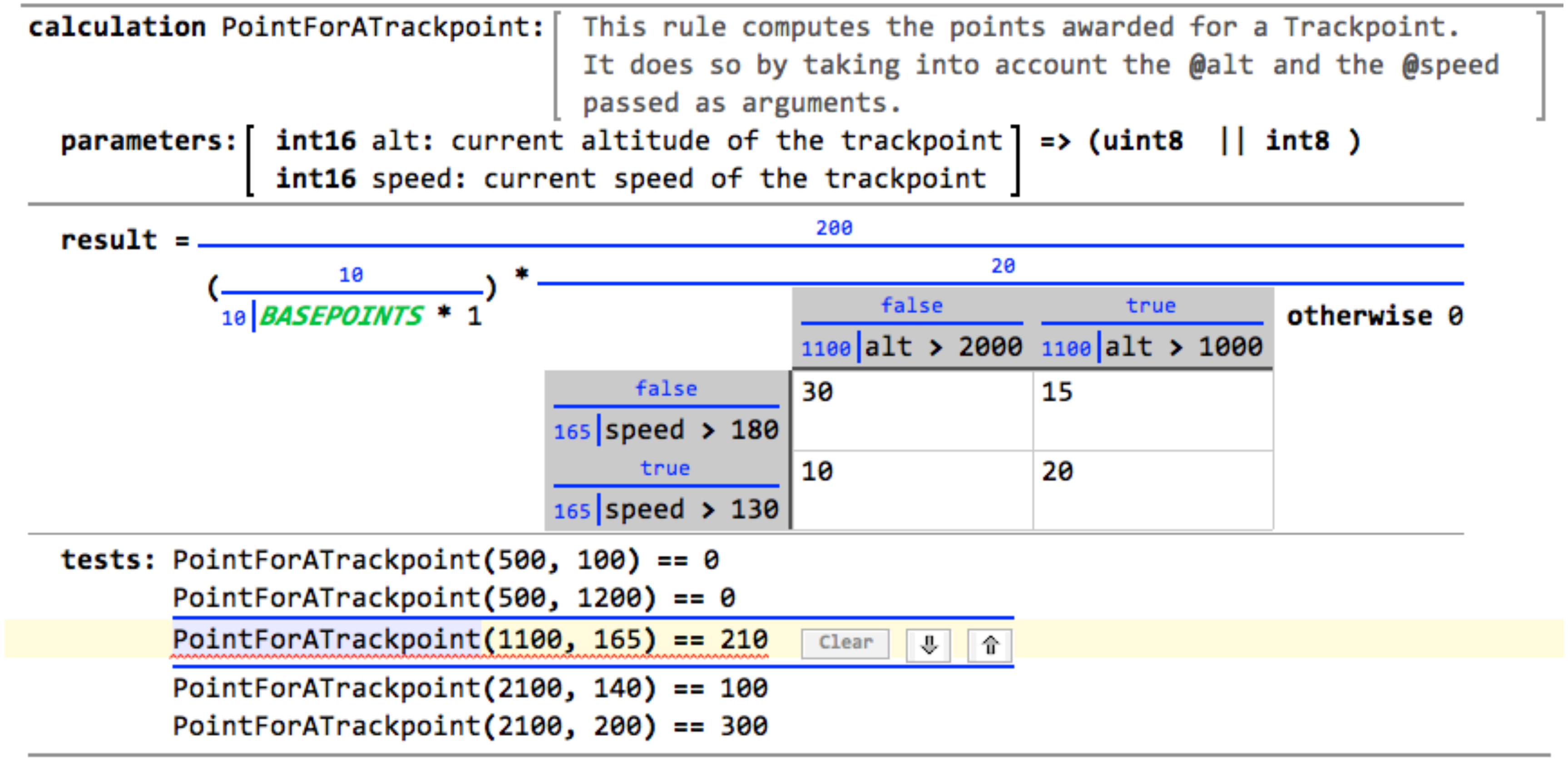
	alt > 2000	alt > 1000	otherwise 0
speed > 180	30	15	
speed > 130	10	20	

```
  te       500, 100) == 0
     Error: failed; expected 210, but was 200
          PointForATrackpoint(500, 1200) == 0
          PointForATrackpoint(1100, 165) == 210
          PointForATrackpoint(2100, 140) == 100
          PointForATrackpoint(2100, 200) == 300
```

Live (interpreted) Business Rules can be Embedded in Req.

Features

Debugging Business Rules („Live Program‘g“)



```
calculation PointForATrackpoint: This rule computes the points awarded for a Trackpoint.
                                 It does so by taking into account the @alt and the @speed
                                 passed as arguments.
  parameters: int16 alt: current altitude of the trackpoint  => (uint8 || int8 )
              int16 speed: current speed of the trackpoint

  result =                         200
        (       10       ) *                      20
         10|BASEPOINTS * 1
```

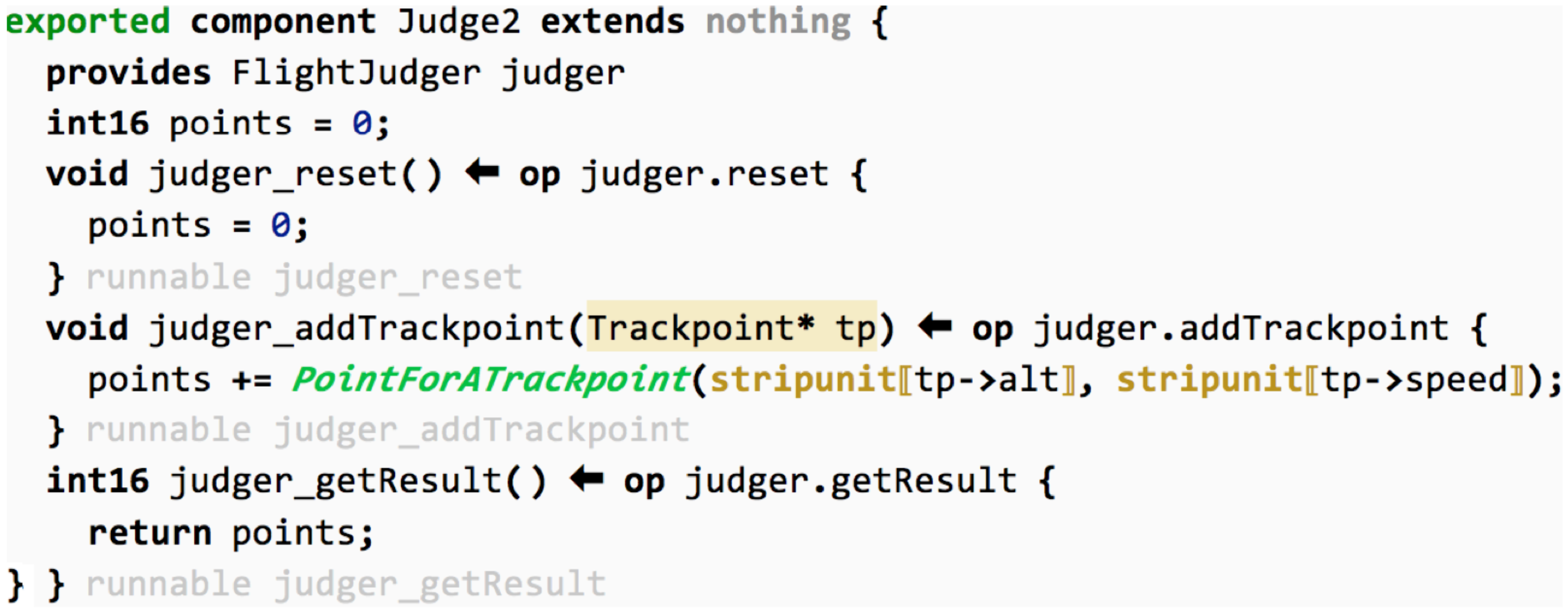
	false 1100|alt > 2000	true 1100|alt > 1000	otherwise 0
false 165|speed > 180	30	15	
true 165|speed > 130	10	20	

```
  tests: PointForATrackpoint(500, 100) == 0
         PointForATrackpoint(500, 1200) == 0
         PointForATrackpoint(1100, 165) == 210   Clear ⇩ ⇧
         PointForATrackpoint(2100, 140) == 100
         PointForATrackpoint(2100, 200) == 300
```

All intermediate expression values shown inline.

Features

Code Referencing Business Rules

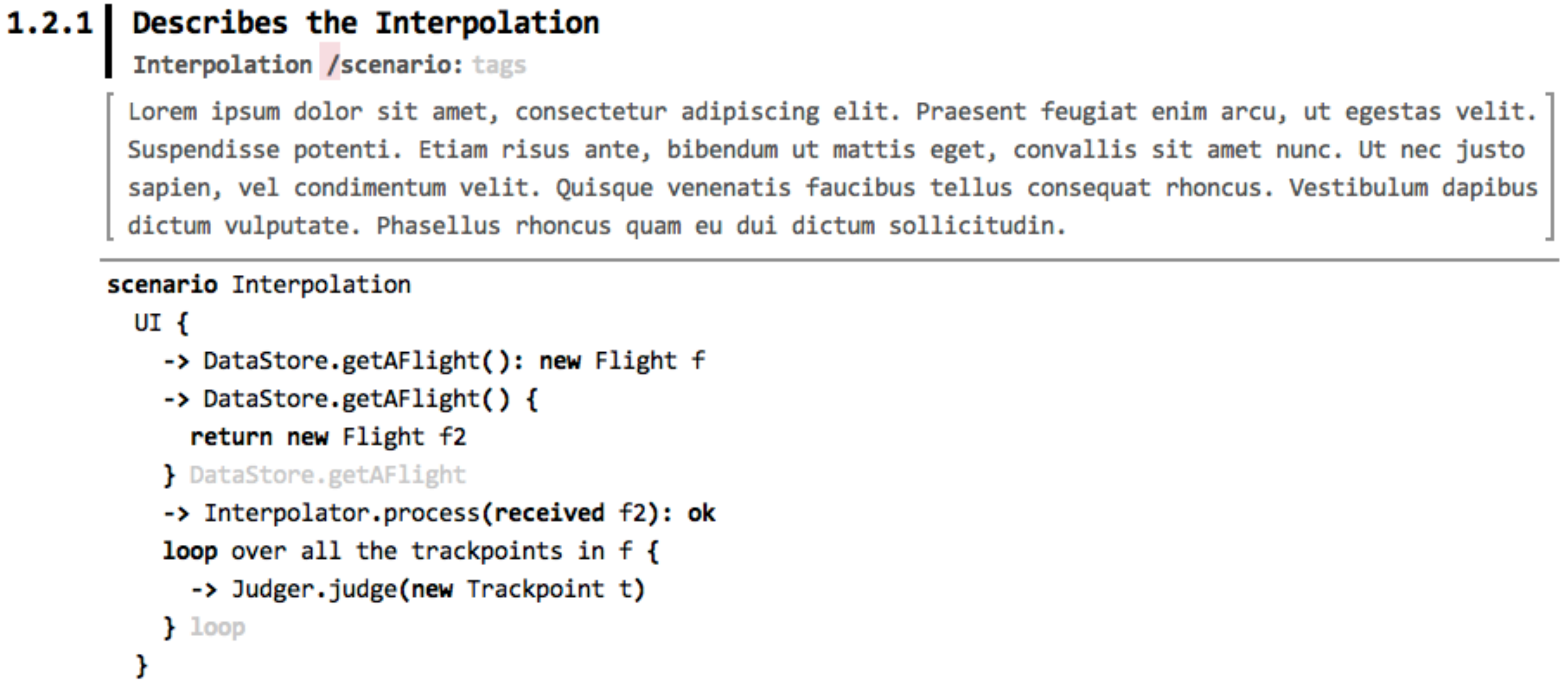


```
exported component Judge2 extends nothing {
  provides FlightJudger judger
  int16 points = 0;
  void judger_reset() ⬅ op judger.reset {
    points = 0;
  } runnable judger_reset
  void judger_addTrackpoint(Trackpoint* tp) ⬅ op judger.addTrackpoint {
    points += PointForATrackpoint(stripunit⟦tp->alt⟧, stripunit⟦tp->speed⟧);
  } runnable judger_addTrackpoint
  int16 judger_getResult() ⬅ op judger.getResult {
    return points;
} } runnable judger_getResult
```

These Business Rules can be „called“ from C Code

Features

Requirements with Scenarios



1.2.1 Describes the Interpolation

Interpolation /scenario: tags

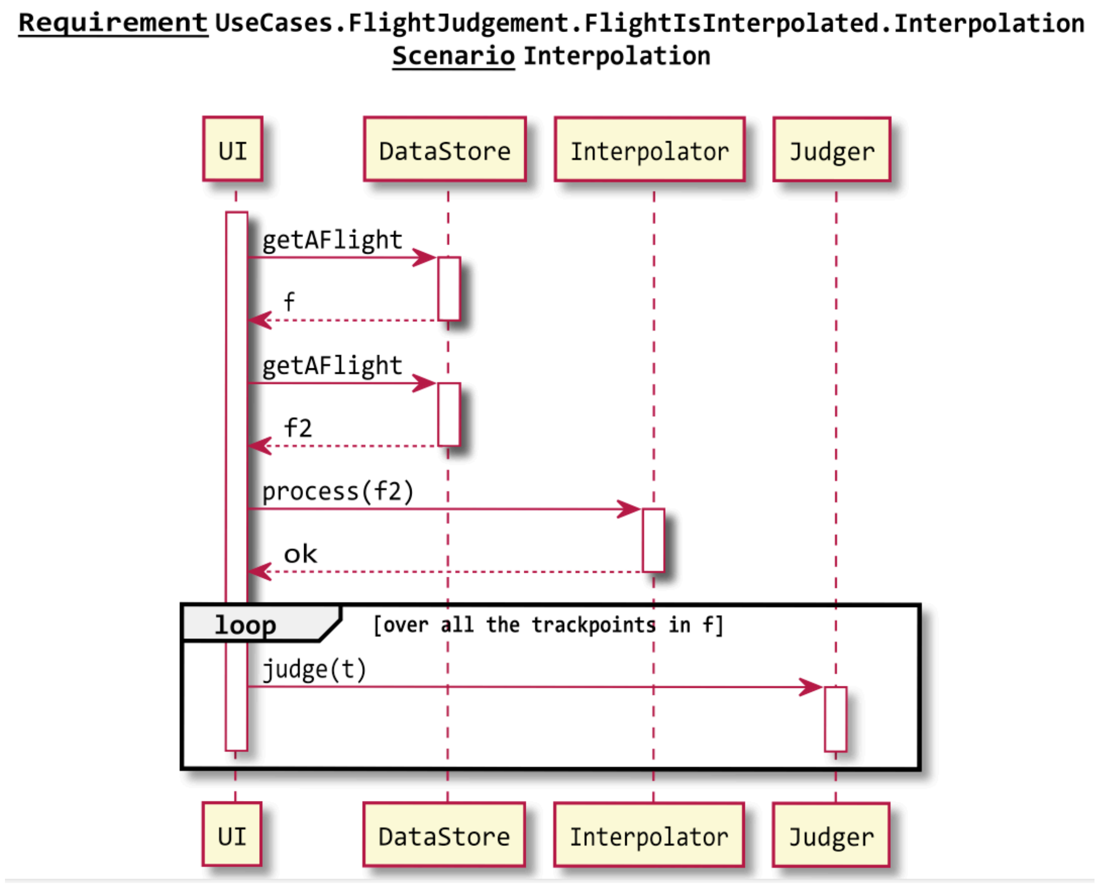
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Praesent feugiat enim arcu, ut egestas velit. Suspendisse potenti. Etiam risus ante, bibendum ut mattis eget, convallis sit amet nunc. Ut nec justo sapien, vel condimentum velit. Quisque venenatis faucibus tellus consequat rhoncus. Vestibulum dapibus dictum vulputate. Phasellus rhoncus quam eu dui dictum sollicitudin.

```
scenario Interpolation
  UI {
    -> DataStore.getAFlight(): new Flight f
    -> DataStore.getAFlight() {
      return new Flight f2
    } DataStore.getAFlight
    -> Interpolator.process(received f2): ok
    loop over all the trackpoints in f {
      -> Judger.judge(new Trackpoint t)
    } loop
  }
```

Requirements are Extensible, e.g. with Scenarios

Features

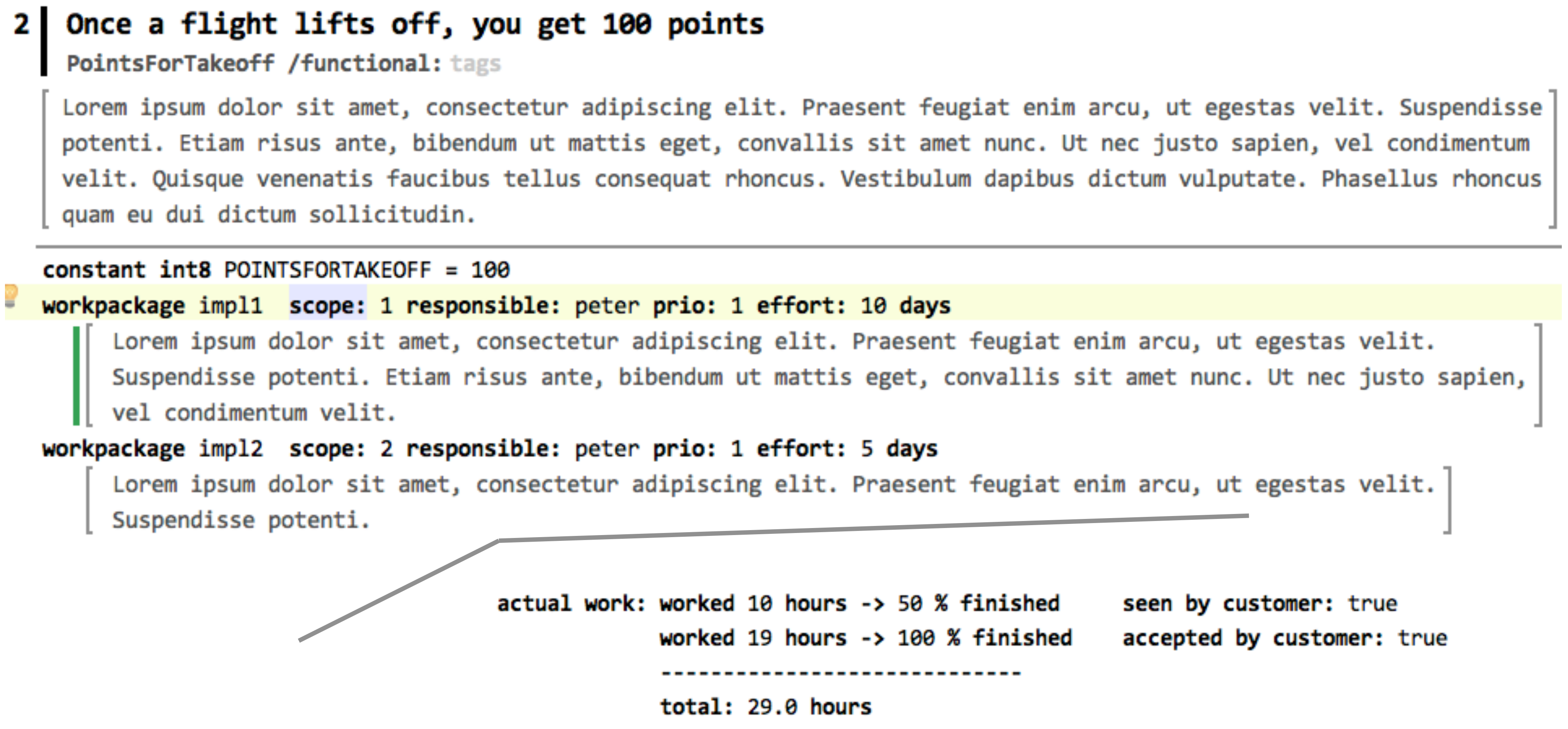
Graphical Scenarios



Scenarios can be Visualized

Features

Workpackages

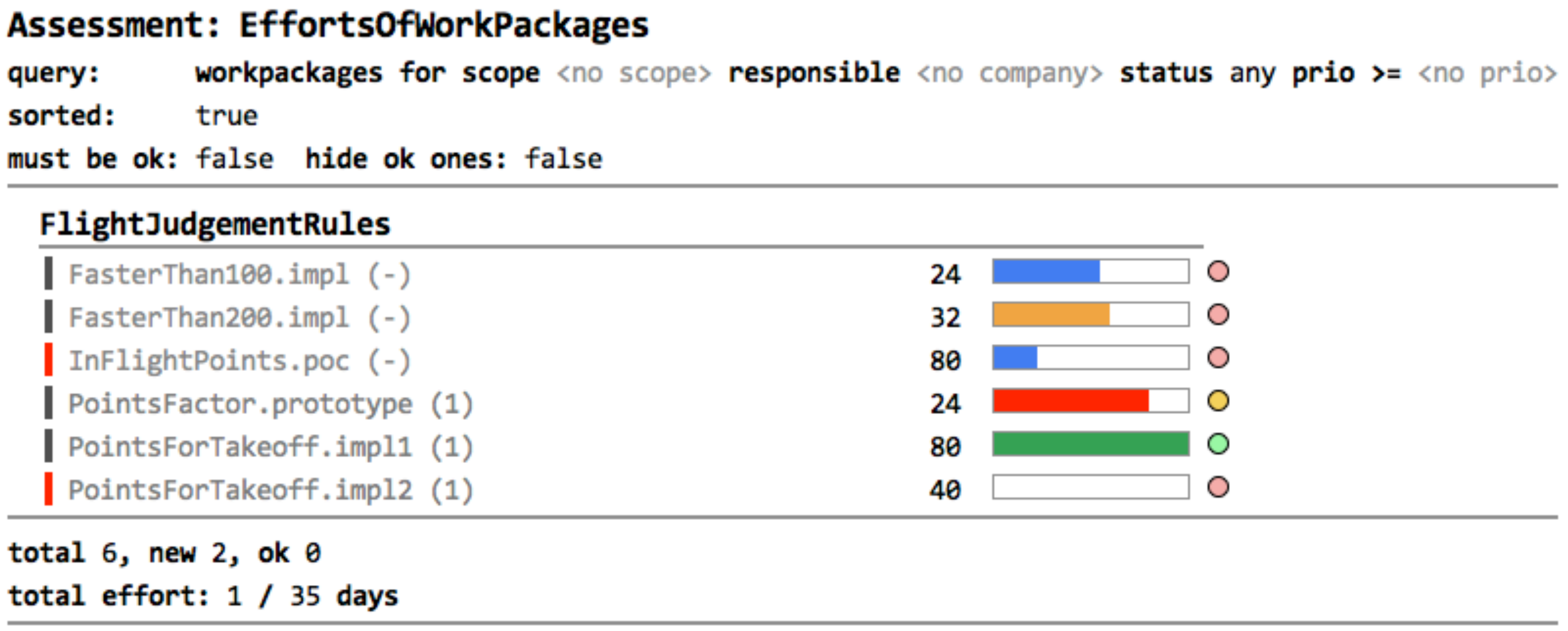


```
2 | Once a flight lifts off, you get 100 points
  | PointsForTakeoff /functional: tags
  [ Lorem ipsum dolor sit amet, consectetur adipiscing elit. Praesent feugiat enim arcu, ut egestas velit. Suspendisse
    potenti. Etiam risus ante, bibendum ut mattis eget, convallis sit amet nunc. Ut nec justo sapien, vel condimentum
    velit. Quisque venenatis faucibus tellus consequat rhoncus. Vestibulum dapibus dictum vulputate. Phasellus rhoncus
    quam eu dui dictum sollicitudin. ]

  constant int8 POINTSFORTAKEOFF = 100
  workpackage impl1  scope: 1 responsible: peter prio: 1 effort: 10 days
      [ Lorem ipsum dolor sit amet, consectetur adipiscing elit. Praesent feugiat enim arcu, ut egestas velit.
        Suspendisse potenti. Etiam risus ante, bibendum ut mattis eget, convallis sit amet nunc. Ut nec justo sapien,
        vel condimentum velit. ]
  workpackage impl2  scope: 2 responsible: peter prio: 1 effort: 5 days
      [ Lorem ipsum dolor sit amet, consectetur adipiscing elit. Praesent feugiat enim arcu, ut egestas velit.
        Suspendisse potenti. ]

                      actual work: worked 10 hours -> 50 % finished      seen by customer: true
                                   worked 19 hours -> 100 % finished     accepted by customer: true
                                   -----------------------------
                                   total: 29.0 hours
```

An extension supports workpackages for requirements.

Features

Workpackage Assessments

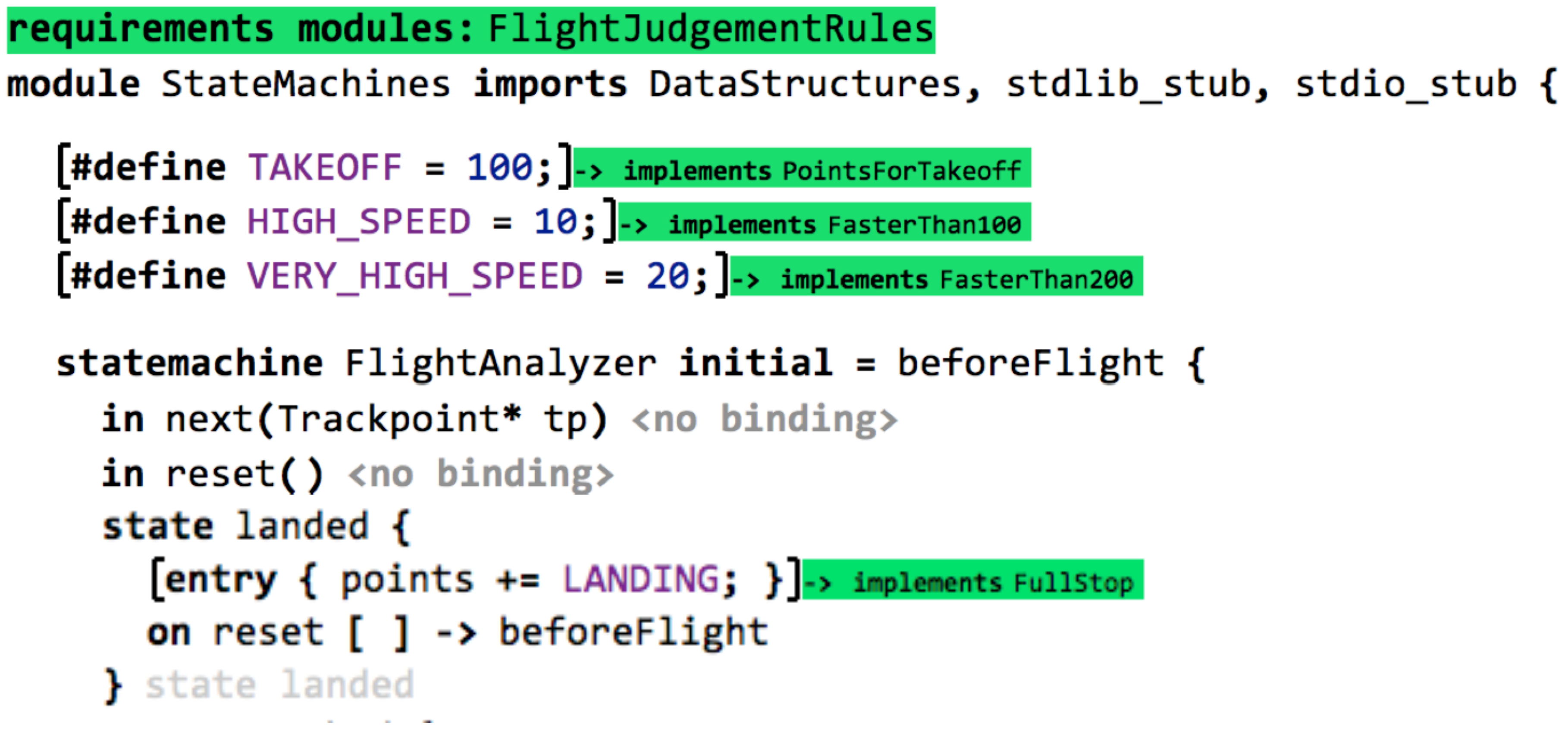


```
Assessment: EffortsOfWorkPackages
query:       workpackages for scope <no scope> responsible <no company> status any prio >= <no prio>
sorted:      true
must be ok: false  hide ok ones: false

  FlightJudgementRules
  | FasterThan100.impl (-)                        24
  | FasterThan200.impl (-)                        32
  | InFlightPoints.poc (-)                        80
  | PointsFactor.prototype (1)                    24
  | PointsForTakeoff.impl1 (1)                    80
  | PointsForTakeoff.impl2 (1)                    40

total 6, new 2, ok 0
total effort: 1 / 35 days
```

A report over the workpackages and the spent work

Features

Requirements Tracing



```
requirements modules: FlightJudgementRules
module StateMachines imports DataStructures, stdlib_stub, stdio_stub {

  [#define TAKEOFF = 100;]-> implements PointsForTakeoff
  [#define HIGH_SPEED = 10;]-> implements FasterThan100
  [#define VERY_HIGH_SPEED = 20;]-> implements FasterThan200

  statemachine FlightAnalyzer initial = beforeFlight {
    in next(Trackpoint* tp) <no binding>
    in reset() <no binding>
    state landed {
      [entry { points += LANDING; }]-> implements FullStop
      on reset [ ] -> beforeFlight
    } state landed
```

Ubiquitous Tracing from Arbitrary Program Elements

5

Example III: Insurance

Insurance Workbench

More Form-Like Notation

```
Rule Set Type DemoRuleSetType

Business objects
person : Person
policy Policy :

Variables:                       Parent
PRMI      : int                  <no parent>
FR        : int
NN        : int
TT        : int                  Libraries
J         : int                  Standard
A3        : int                  Extra
G3        : int
ANUI      : int
X         : int
```

This workbench is to be used by insurance experts

Insurance Workbench

More Form-Like Notation

Rule Set Type DemoRuleSetType

Business objects

person : **Person**
policy **Policy** :

Variables:

PRMI	:	int
FR	:	int
NN	:	int
TT	:	int
J	:	int
A3	:	int
G3	:	int
ANUI	:	int
X	:	int

Parent

<no parent>

Libraries

Standard
Extra

Rule Set Type DemoRuleSetType

Business objects

Variables:

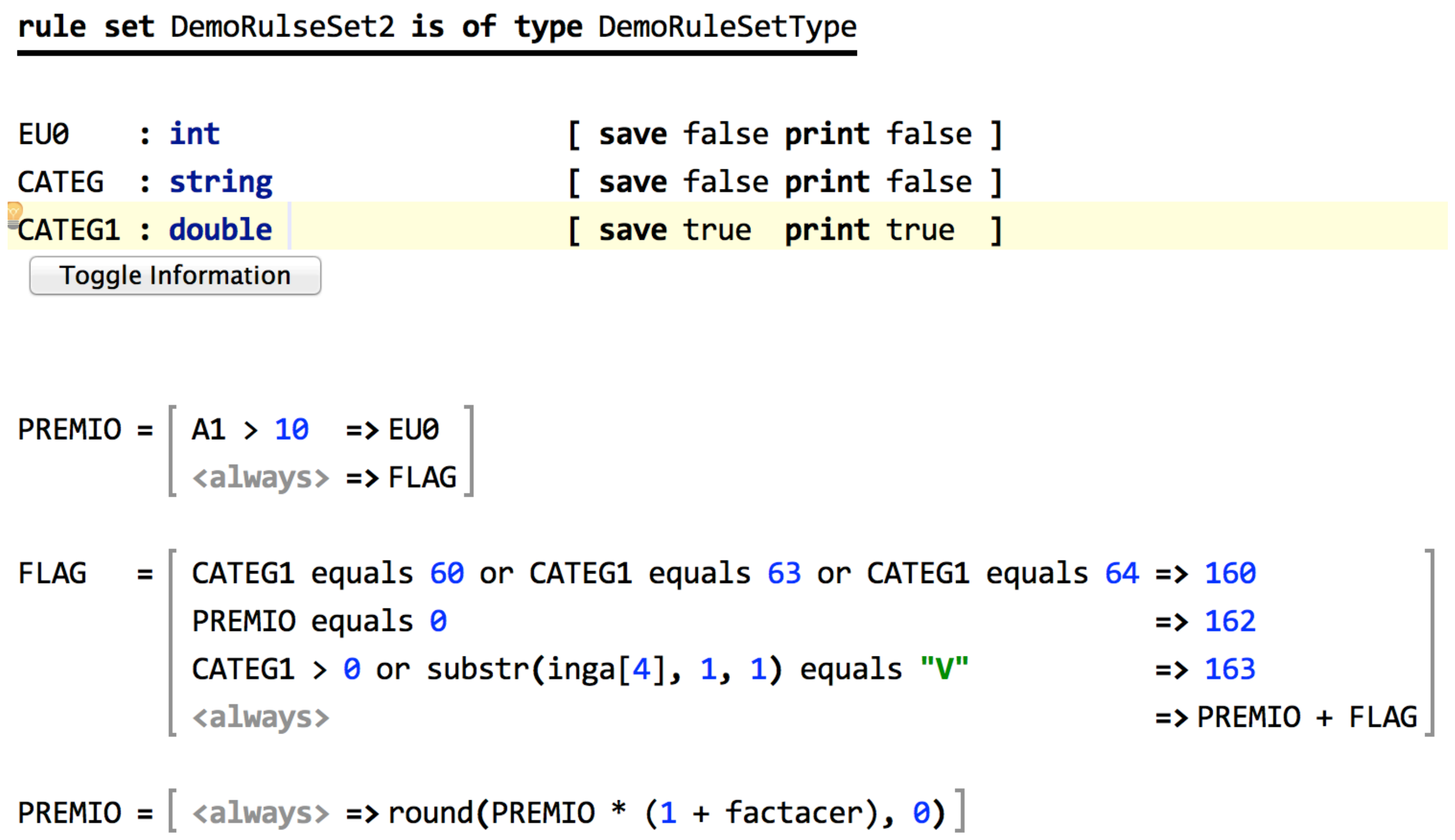
Parent

Libraries

This workbench is to be used by insurance experts

Insurance Workbench

More Form-Like Notation – with Expressions



```
rule set DemoRulseSet2 is of type DemoRuleSetType

EU0    : int                    [ save false print false ]
CATEG  : string                 [ save false print false ]
CATEG1 : double                 [ save true  print true  ]
 Toggle Information


PREMIO = [ A1 > 10   => EU0  ]
         [ <always> => FLAG ]

FLAG   = [ CATEG1 equals 60 or CATEG1 equals 63 or CATEG1 equals 64 => 160          ]
         [ PREMIO equals 0                                          => 162          ]
         [ CATEG1 > 0 or substr(inga[4], 1, 1) equals "V"           => 163          ]
         [ <always>                                                 => PREMIO + FLAG ]

PREMIO = [ <always> => round(PREMIO * (1 + factacer), 0) ]
```

Non-Programmers like Forms and Buttons – and need Lang's

Insurance Workbench

Mathematical Notation

$$\texttt{int other(a : int, b : int) ==>}\; a + b + \sum_{i=1}^{5}\left[\, i \,\right] + \prod_{p=1}^{3}\left[\, p \,\right]$$

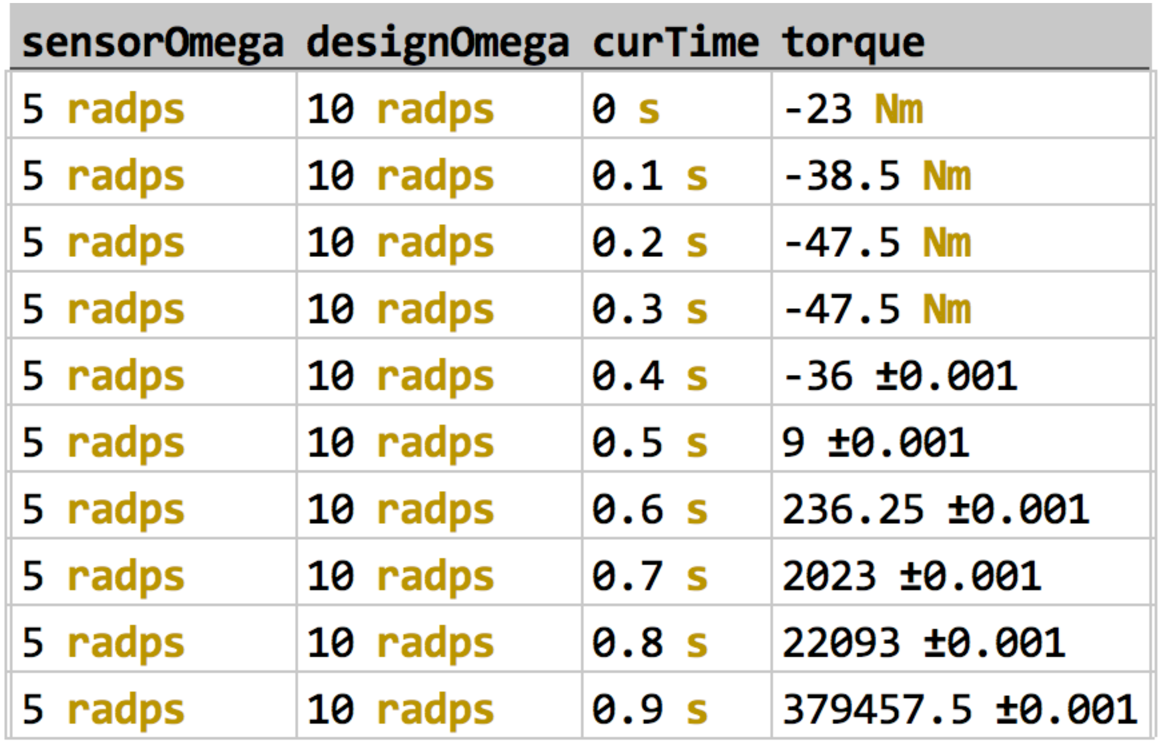
$$\texttt{local} = \left[\; \texttt{A1 =>} \left(\frac{\sum_{i=1}^{NN}\left[(D(X + ANUI + i - 1) - D(X + ANUI + i)) * \left(1 - \frac{TM18[i]}{TM17}\right) \right]}{D(X + ANUI)} \right) \right]$$

$$\texttt{int rate(age : int) ==>}\; 1 + \frac{1 + ANUI + \frac{age}{AOPS - 9}}{4 * 5 + \sum_{i=8}^{12}\left[\, i * 8 \,\right]} + in01$$

This workbench was used by insurance domain experts

Insurance Workbench

Tables (taken from diff. Example)

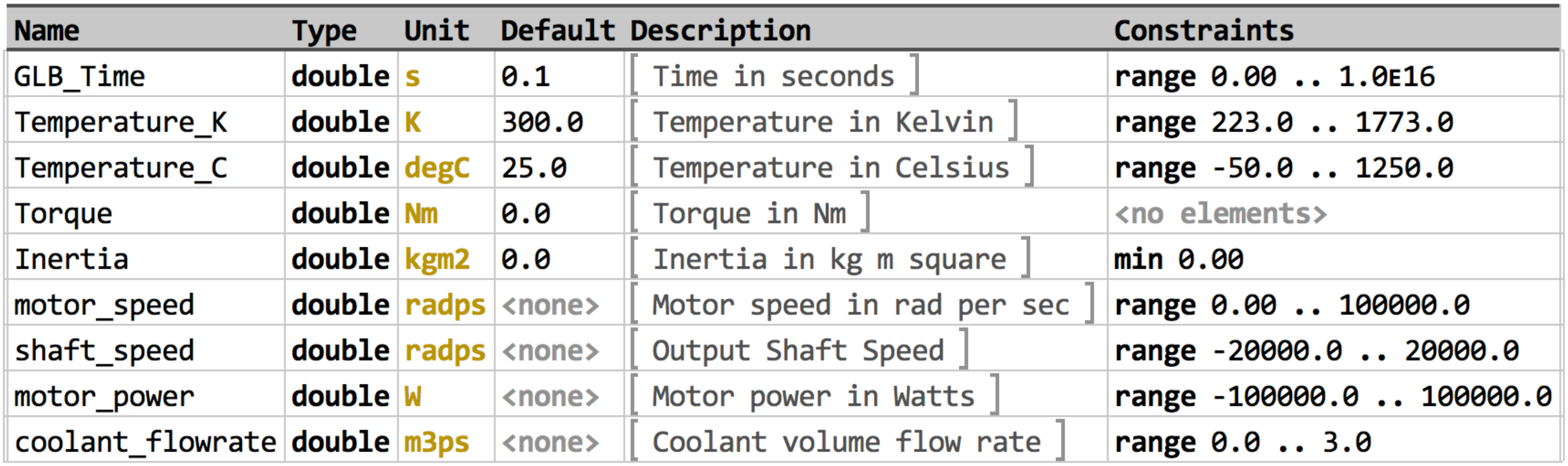


sensorOmega	designOmega	curTime	torque
5 radps	10 radps	0 s	-23 Nm
5 radps	10 radps	0.1 s	-38.5 Nm
5 radps	10 radps	0.2 s	-47.5 Nm
5 radps	10 radps	0.3 s	-47.5 Nm
5 radps	10 radps	0.4 s	-36 ±0.001
5 radps	10 radps	0.5 s	9 ±0.001
5 radps	10 radps	0.6 s	236.25 ±0.001
5 radps	10 radps	0.7 s	2023 ±0.001
5 radps	10 radps	0.8 s	22093 ±0.001
5 radps	10 radps	0.9 s	379457.5 ±0.001

A bit like „Excel“ with a real language behind it.

Insurance Workbench

Tables (taken from diff. Example)



Name	Type	Unit	Default	Description	Constraints
GLB_Time	**double**	s	0.1	Time in seconds	**range** 0.00 .. 1.0E16
Temperature_K	**double**	K	300.0	Temperature in Kelvin	**range** 223.0 .. 1773.0
Temperature_C	**double**	degC	25.0	Temperature in Celsius	**range** -50.0 .. 1250.0
Torque	**double**	Nm	0.0	Torque in Nm	<no elements>
Inertia	**double**	kgm2	0.0	Inertia in kg m square	**min** 0.00
motor_speed	**double**	radps	<none>	Motor speed in rad per sec	**range** 0.00 .. 100000.0
shaft_speed	**double**	radps	<none>	Output Shaft Speed	**range** -20000.0 .. 20000.0
motor_power	**double**	W	<none>	Motor power in Watts	**range** -100000.0 .. 100000.0
coolant_flowrate	**double**	m3ps	<none>	Coolant volume flow rate	**range** 0.0 .. 3.0

A bit like „Excel" with a real language behind it.

6

Summing up

Summing Up

Key Points

To build meaningful tools, the data must be extended.

Extending the tool (buttons, views, ...) is not enough!

Summing Up

Key Points

Structured Data can be expressed with languages.

Languages are data formats plus syntax and IDE.

Summing Up

Key Points

Language Engineering supports extension and composition

This supports adapting tools for specific domains easily.

Summing Up

Key Points

IDE-style tools are very good for editing data/programs.

We've got a lot of experience from regular programming.

Summing Up

Key Points

Language Workbenches are the key enabling technology.

MPS is IMHO the most powerful, but it's not the only one!

Summing Up

Key Points

Let`s build new classes of tools!

... which make meaningful extensibility a reality!

The End.

DSL Engineering

Designing, Implementing and Using Domain-Specific Languages

Markus Voelter

with Sebastian Benz, Christian Dietrich, Birgit Engelmann, Mats Helander, Lennart Kats, Eelco Visser, Guido Wachsmuth

www.dslbook.org

voelter.de
dslbook.org
mbeddr.com
jetbrains.com/mps

The End.