Generic Tools - Specific Languages

On the Art of Building Software Development Tools

Markus Voelter
independent/itemis

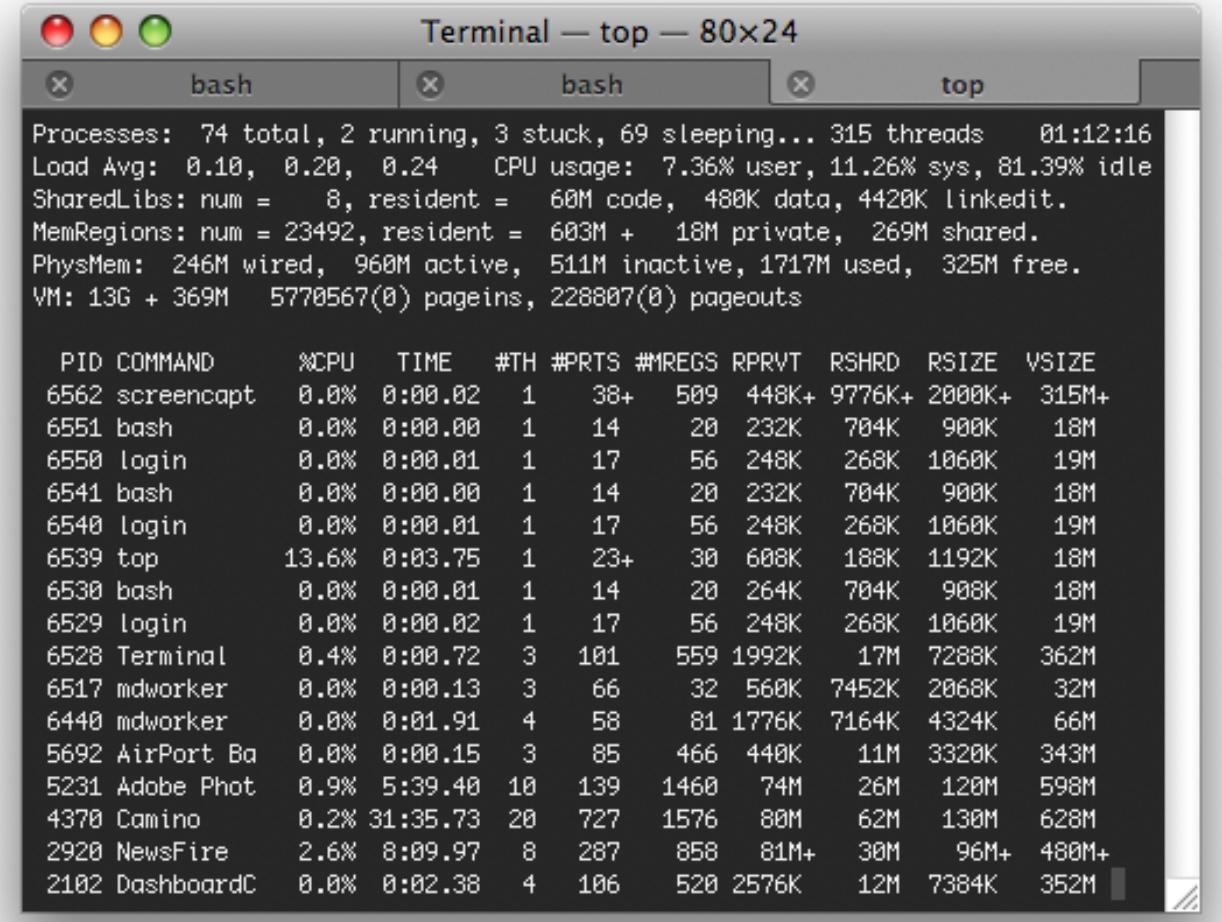
http://voelter.de
voelter@acm.org
@markusvoelter

1. Tools
2. Extensibility
3. Challenges
4. GTSL
5. An Example
6. Wrap Up

Tools

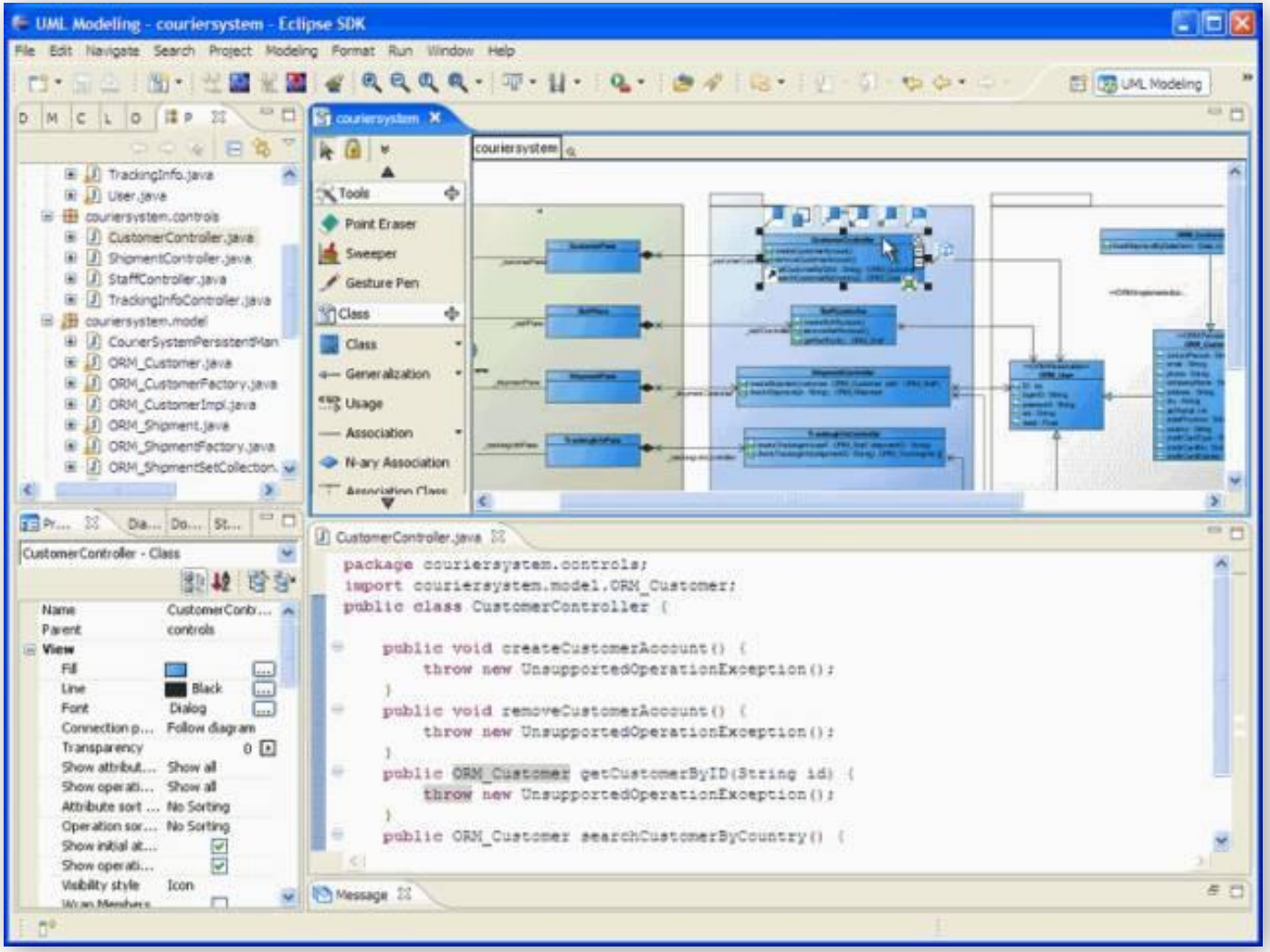
Tools

Command-Line Tools



Tools

UI Tools



Tool

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.

Tool Extensibility

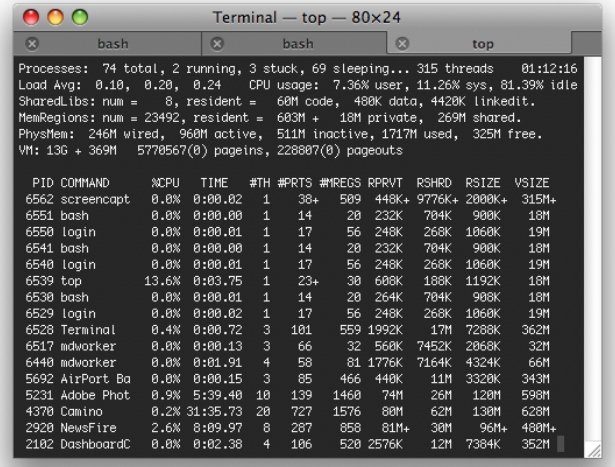
Study Findings III

> 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

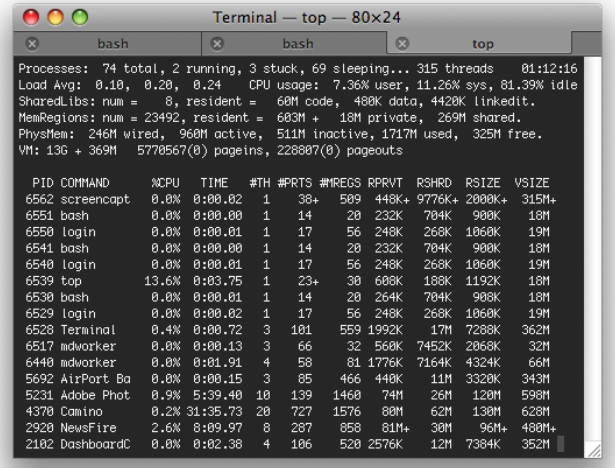


New File Formats

New Processors

Tool Extensibility

Command-Line Tools



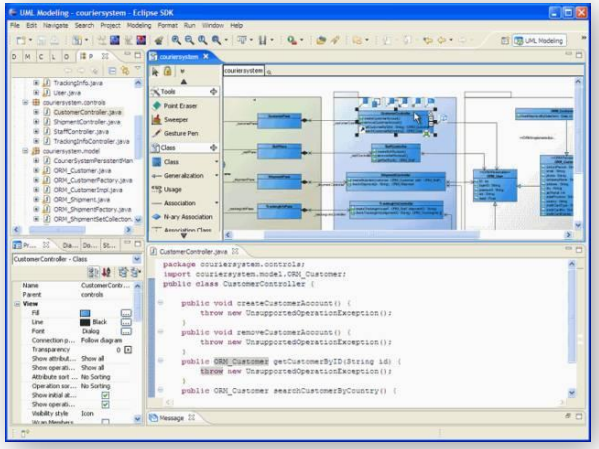
New File Formats

New Processors

Assemble Components (Pipes & Filters)

Tool Extensibility

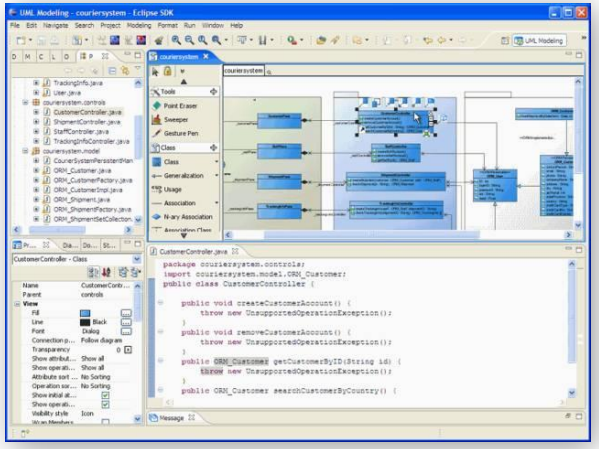
UI Tools



Buttons Views
Menus Actions

Tool Extensibility

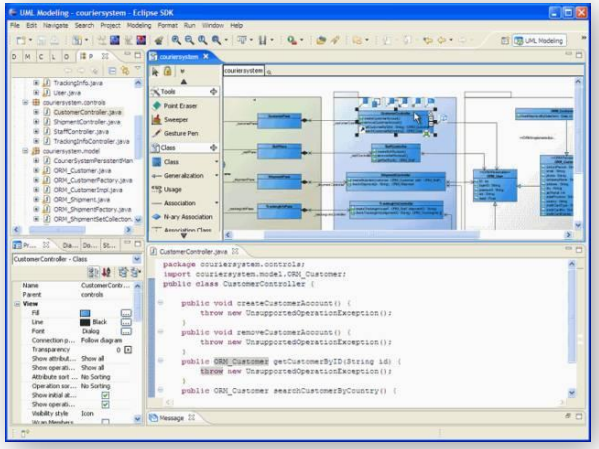
UI Tools



Buttons Views
Menus Actions
New Languages
New Editors

Tool Extensibility

UI Tools



Buttons Views
Menus Actions
New Languages
New Editors

Platform/Plugin Systems

Challenges

Overview

Context: embedded programming

Examples are from Embedde
Programming and use C as th
„data format“

But applies similarly to other
or other data formats/languag

Extensibility Examples

Physical Units: The Challenge

How do you work with
physical units in your

Extensibility Examples

Physical Units: The Challenge

```
// in file example.c
int distance = 10;
int time     = 1;
int speed    = distance / time;
```

Extensibility Examples

Physical Units: The Challenge

```
// in file example.c
int distance = 10;
int time     = 1;
int speed    = distance / time;

int speed  = time / distance;
```

How do you detect this e

Extensibility Examples

Physical Units: The Challenge

```
int speed  = time / distance;
```

How do you detect this er

You need to:

- Tool (to do the checking)
- Data (the units in the code)

Extensibility Examples

Physical Units via Comments

```
int/*#m*/ distance  = 10 /*#m*/;
int/*#s*/ time      = 1 /*#s*/;
int/*#mps*/ speed   = distance / time;
```

Bad

Extensibility Examples

Physical Units via Macros

```
UT(int, m) distance = UV(10, s);
UT(int, s) time     = UV(1, s);
UT(int, mps) speed  = distance / time;
```

Bad

Extensibility Examples

Physical Units via external XML

```
<unitdeclarations>
  <unit name="m" for="distance"/>
  <unit name="s" for="time"/>
  <unit name="mps" for="speed" calculateAs="m/s"/>
</unitdeclarations>
<programmarkup>
  <globalvar file="example.c" name="distance" unit="m"/>
  <globalvar file="example.c" name="time" unit="s"/>
  <globalvar file="example.c" name="speed" unit="mps"/>
</programmarkup>
```

Bad

Extensibility Examples

Physical Units via Extension

```
int/m/ distance  = 10 m;
int/s/ time      = 1 s;
int/mps/ speed   = distance / time;
```

Gooc

Extensibility Examples

Physical Units via Extension

```
int/m/ distance  = 10 m;
int/s/ time      = 1 s;
int/mps/ speed   = distance / time;
```

You get

- Type Checker (to do the checking)
- Program Code (the units in the code)

Gooo

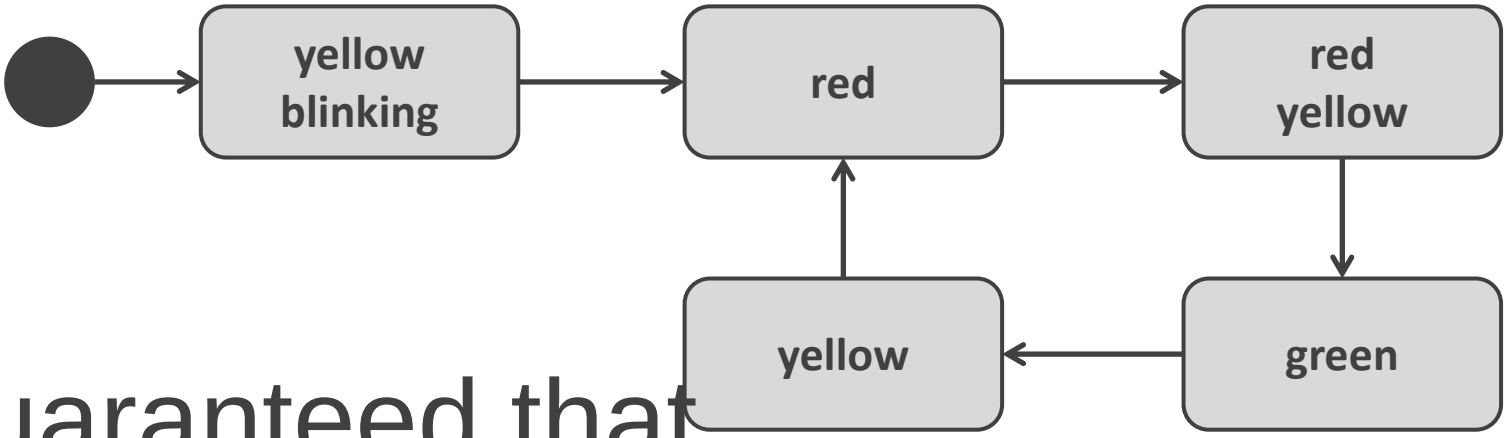
Extensibility Examples

State Machines: The Challenge

How do you represent
state-based behavior i
and support analyses?

Extensibility Examples

State Machines: The Challenge



Is it guaranteed that ...

... the TL get green eventually?

... if the TL is turned off/on, it starts in

... the TL never goes from yellow to g

Extensibility Examples

State Machines via C idioms

```
// a state machine that transitions into S2
// when E1 is received while the machine is in S1
void execute_StateMachine( Event_Enum evt ) {
  switch (currentState) {
    case S1: switch (evt) {
               case E1: if ( guard for E1 in S1 ) {
                          // execute exit actions for S1
                          currentState = S2;
                          // execute entry actions for S2
                          break;
                        }
             }
    case S2: ...
    ...
  }
}
```

Bad

Extensibility Examples

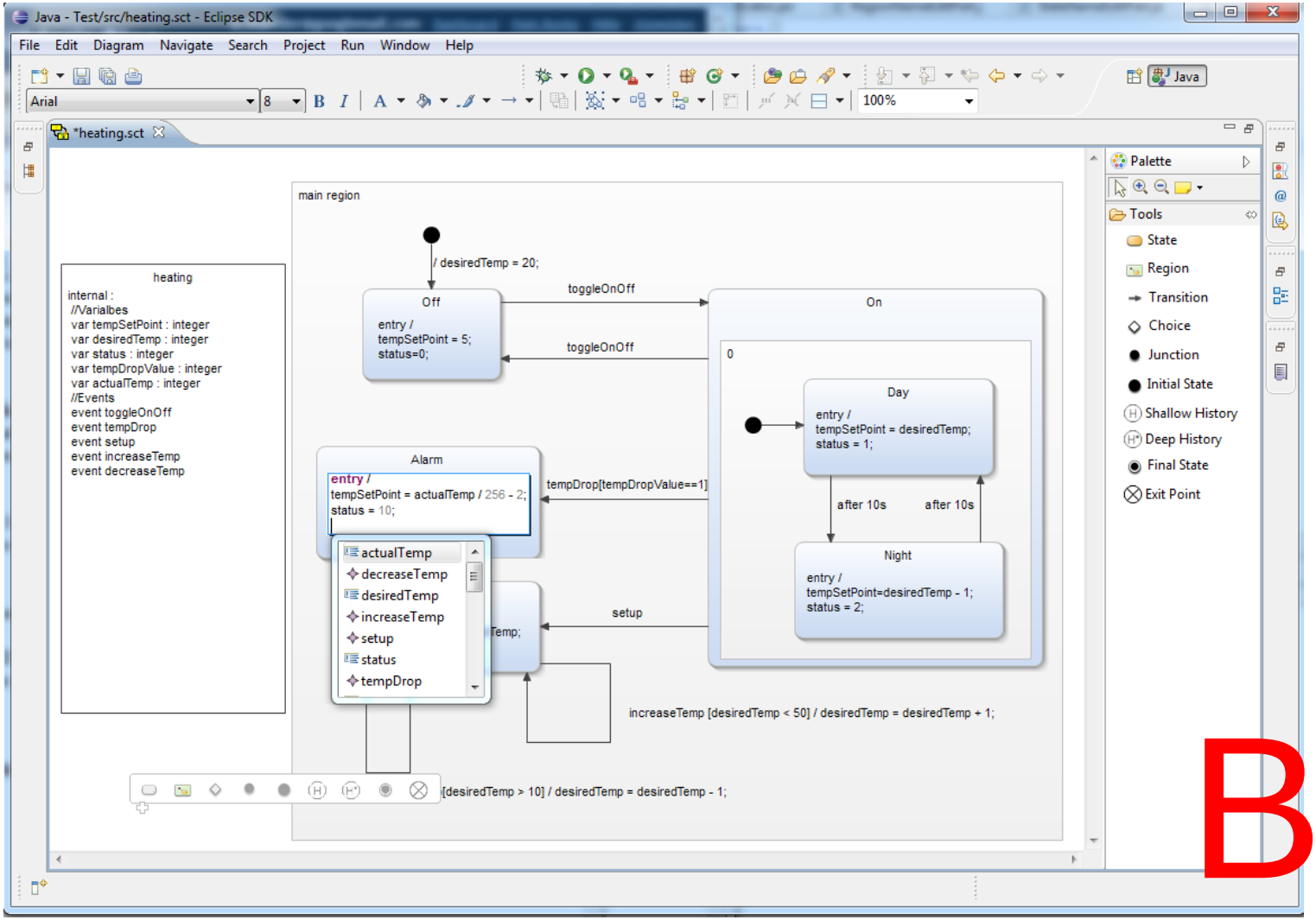
State Machines via C idioms

```
// a state machine that transitions into S2
// when E1 is received while the machine is in S1
// -1 means "do nothing".
                              //  S1  S2  S3 ...
int[N_EVT][N_STATE] =  { {     1, -1, -1 }     // E1
                         {    -1, -1, -1 }     // E2
                         ...
                                              };
```

Bad

Extensibility Examples

State Machines via External Tool



Bad

Extensibility Examples

State Machines: The Challenge

How do you perform anlyses on state machi

You need to:

- Tool (to do the checking)
- Data („clean“ state machines)

Extensibility Examples

State Machines via Extensions

```
statemachine SM {
  event E1
  state S1 {
    entry { // entry action for S1 }
    on E1 [guard for E1 in S1] -> S2
    exit { // exit action for S1 }
  }
  state S2 {
    ...
  }
  ...
}
```

Gooc

Extensibility Examples

State Machines via Extensions

```
statemachine SM {
  event E1
  state S1 {
    entry { // entry action for S1 }
    on E1 [guard for E1 in S1] -> S2
    exit { // exit action for S1 }
  }
  state S2 {
    ...
  }
  ...
}
```

You get:

- Constraint Checker (to do the checking)
- Program Code (the units in the code)

Good

Extensibility Examples

Tracing: The Challenge

How do you add trace
requirements anywhe
your code, robustly?

Extensibility Examples

Tracing via Macros

```
TRACE(REQ_CALIBRATION)
int calibrate( int measurement ) {
  return measurement * FACTOR + OFFSET;
}

int getValue() {
  int raw = readFromDriver(ADC1_ADDRESS);
  TRACE(REQ_CALIBRATION)
  return calibrate(raw);
}
```

Bad

Extensibility Examples

Tracing via Macros

```
TRACE(REQ_CALIBRATION)
int calibrate( int measurement ) {
  return measurement * FACTOR + OFFSET;
}

int getValue() {
  int raw = readFromDriver(ADC1_ADDRESS);
  TRACE(REQ_CALIBRATION)
  return calibrate(raw);
}
```

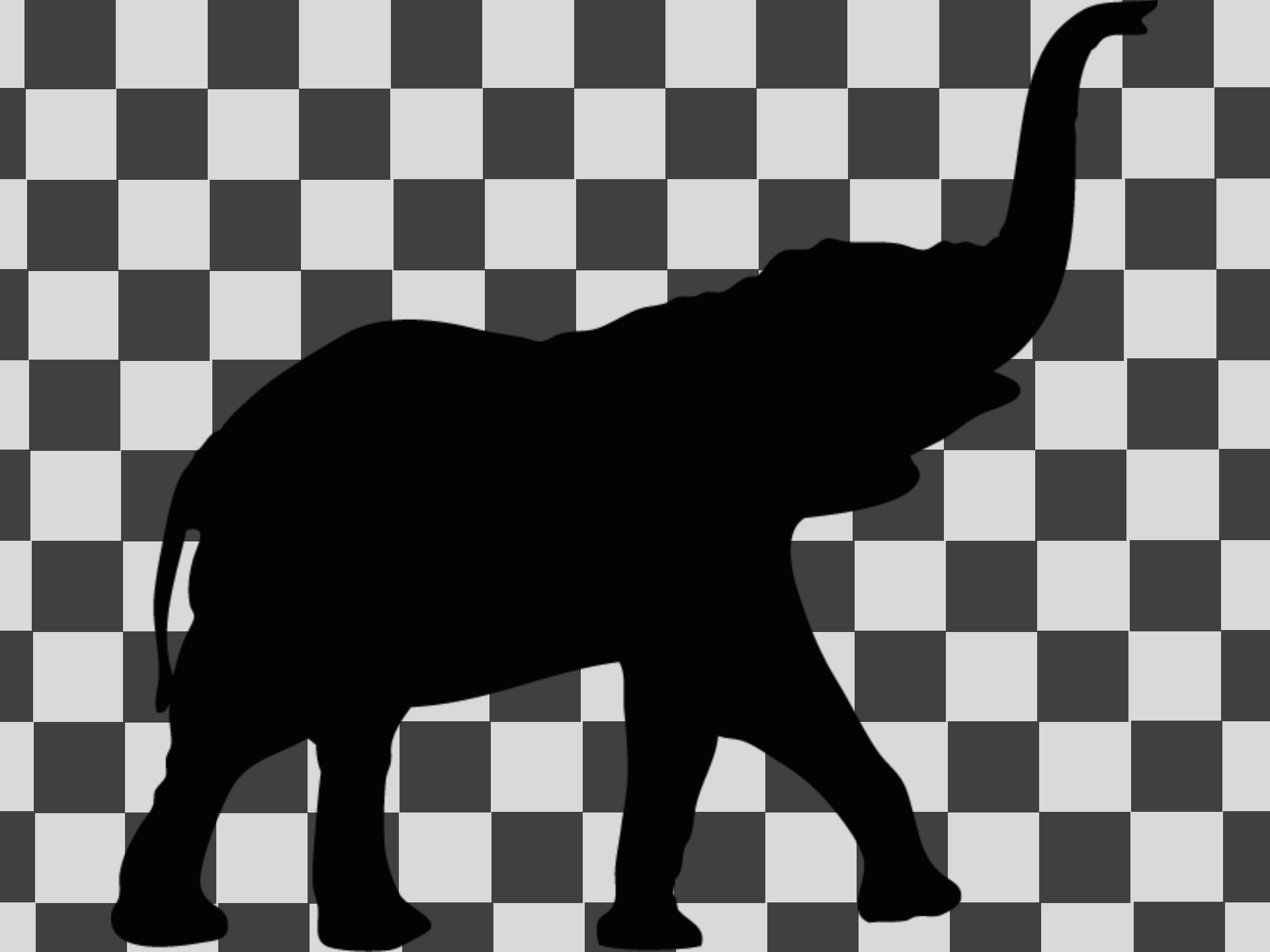
You need:

- Tooling (to create trace reports)
- Data (robust trace annotations)

Extensibility Examples

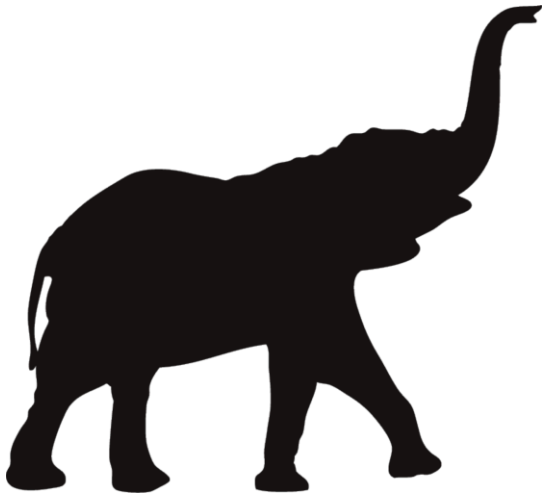
Tracing via Language Extensions

You get the idea :-)



Extensibility Examples

Combinations

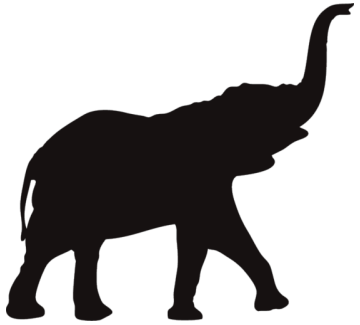


How do you combine these (and other) extenions?

Extensibility Examples

Combinations

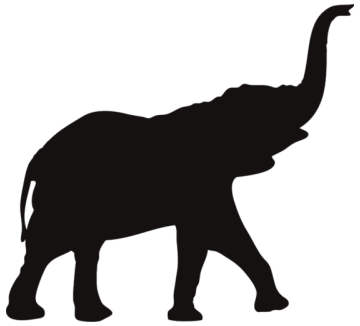
```
statemachine TrainDoorController {
  event DOOR_BUTTON;
  state DOORS_CLOSED {
    trace REQ_BUTTON_OPENS_DOORS_ONLY_OPEN_WHEN_STOPPED
    on DOOR_BUTTON [speed > 0 mps] -> DOORS_OPEN
  }
  state DOORS_OPEN {
    entry { openDoors(); }
    trace REQ_BUTTON_CLOSES_DOORS_WHEN_OPEN
    on DOOR_BUTTON [] -> DOORS_CLOSED
    exit { closeDoors(); }
  }
  ...
}
```



Extensibility Examples

Combinations

```
statemachine TrainDoorController {
  event DOOR_BUTTON;
  state DOORS_CLOSED {
    trace REQ_BUTTON_OPENS_DOORS_ONLY_OPEN_WHEN_STOPPED
    on DOOR_BUTTON [speed > 0 mps] -> DOORS_OPEN
  }
  state DOORS_OPEN {
    entry { openDoors(); }
    trace REQ_BUTTON_CLOSES_DOORS_WHEN_OPEN
    on DOOR_BUTTON [] -> DOORS_CLOSED
    exit { closeDoors(); }
  }
  ...
}
```

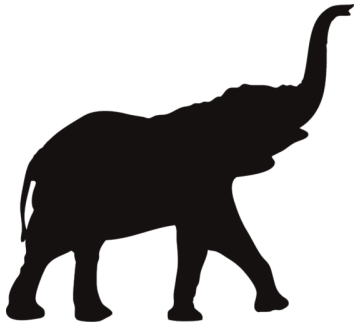


Extensibility Examples

Combinations

- Statemachines

```
statemachine TrainDoorController {
  event DOOR_BUTTON;
  state DOORS_CLOSED {
    trace REQ_BUTTON_OPENS_DOORS_ONLY_OPEN_WHEN_STOPPED
    on DOOR_BUTTON [speed > 0 mps] -> DOORS_OPEN
  }
  state DOORS_OPEN {
    entry { openDoors(); }
    trace REQ_BUTTON_CLOSES_DOORS_WHEN_OPEN
    on DOOR_BUTTON [] -> DOORS_CLOSED
    exit { closeDoors(); }
  }
  ...
}
```



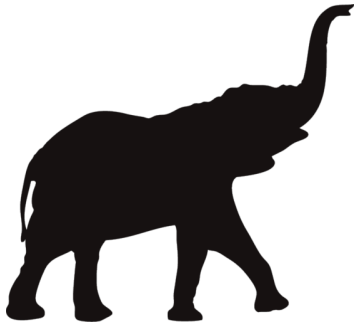
Extensibility Examples

Combinations

Statemachines

```
statemachine TrainDoorController {
  event DOOR_BUTTON;
  state DOORS_CLOSED {
    trace REQ_BUTTON_OPENS_DOORS_ONLY_OPEN_WHEN_STOPPED
    on DOOR_BUTTON [speed > 0 mps] -> DOORS_OPEN
  }
  state DOORS_OPEN {
    entry { openDoors(); }
    trace REQ_BUTTON_CLOSES_DOORS_WHEN_OPEN
    on DOOR_BUTTON [] -> DOORS_CLOSED
    exit { closeDoors(); }
  }
  ...
}
```

Tracin



Extensibility Examples

Combinations

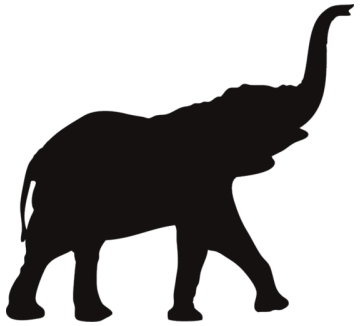
Q

Statemachines

```
statemachine TrainDoorController {
  event DOOR_BUTTON;
  state DOORS_CLOSED {
    trace REQ_BUTTON_OPENS_DOORS_ONLY_OPEN_WHEN_STOPPED
    on DOOR_BUTTON [speed > 0 mps] -> DOORS_OPEN
  }
  state DOORS_OPEN {
    entry { openDoors(); }
    trace REQ_BUTTON_CLOSES_DOORS_WHEN_OPEN
    on DOOR_BUTTON [] -> DOORS_CLOSED
    exit { closeDoors(); }
  }
  ...
}
```

Units

Tracin



Extensibility Examples

Combinations (in an actual tool)

```
[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).]
statemachine FlightAnalyzer initial = beforeFlight {
  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; }]-> implements PointsForTakeoff
  } state beforeFlight
  //[This represents the state in which the airplane flies.
    It has several substates. Note how it uses the @top(VERY_HIGH_SPEED)
    and @top(HIGH_SPEED) constants. These constants are defined in the
    same module @module(StateMachines).]
  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 && tp->alt == 0 m] -> airborne { points += VERY_HIGH_SPEED; }]-> implements FasterThan200
    [on next [tp->speed > 100 mps && tp->speed <= 200 mps && tp->alt == 0 m] -> airborne
        { points += HIGH_SPEED; }]-> implements FasterThan100
    on reset [ ] -> beforeFlight
  } state airborne
```

Key Takeaway Point

Tool Extension is not enough!

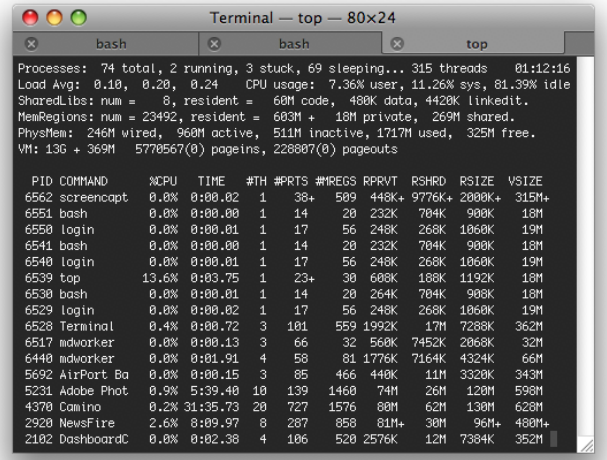
Key Takeaway Point

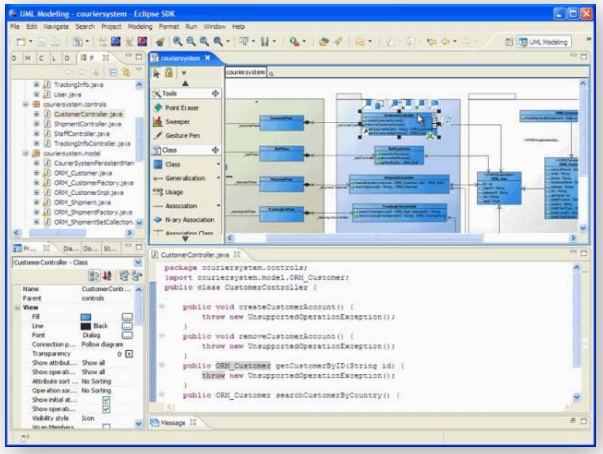
Tool Extension is not enough!

Focus on the data first!

Key Takeaway Point

Tool Extension is not enough!

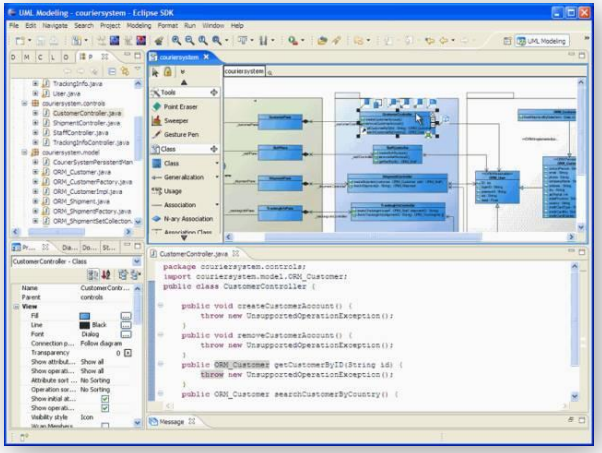




These both do not explicitly support

Key Takeaway Point

Tool Extension is not enough!



Relatively high effort to reimple-ment editors

Key Takeaway Point

Tool Extension is not enough!

Focus on the data first!

Generic Tools

GTSL

Specific Language

Thought Process

From Data Formats To Languages

Thought Process

From Data Formats To Languages

Structure, Constraints, Sema

Data Format

Thought Process

From Data Formats To Languages

Structure, Constraints, Sema

Data Format + Syntax

Language

Thought Process

From Data Formats To Languages

Languages

Thought Process

Language Engineering

Language Reu

Languages

Language Modularization

Language Composition

Thought Process

Language Engineering

Language Reu

Languages

Language

Modularization

Language

Composition

Language Engineering

Thought Process

Language Engineering

Languages

Language Engineering

Thought Process

Language Engineering

Languages

Language Engineering

Text Tables

Math Symbols

Graphics Forms

Thought Process

Language Engineering

Languages

Language Engineering

Text Tables

Math Symbols

Graph Forms

Syntactic Diversity

Thought Process

Language Engineering

Languages

Language Engineering

Syntactic Diversity

Thought Process

Language Workbenches

Languages

Language Engineering

Syntactic Diversity

But does this really work?

Thought Process

Language Workbenches

Languages

Language Engineering

Syntactic Diversity

But does this ~~really work?~~

Language Workbenches

Thought Process

Language Workbenches

Languages

Language Engineering

Syntactic Diversity

Language Workbenches

Generic Tools, Specific Languages

Ingredients

Languages

Language Engineering

Syntactic Diversity

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

Generic Tools

Languages

Language Engineering

Syntactic Diversity

Language Workbenches

(we don‘t have to reimplement editors and synchronizers)

Generic Tools, Specific Languages

Ingredients

Specific Languages

- Languages
- Language Engineering
- Syntactic Diversity

support

Generic Tools

- Language Workbenches

Language Workbenches

Typical Features

Language Workbenches

Typical Features

Language Definition, Reuse, Extension, Composition

Language Workbenches

Typical Features

Language Definition,
Reuse, Extension,
Composition
Mixing Notations

Language Workbenches

Typical Features

Language Definition,
Reuse, Extension,
Composition
Mixing Notations
Type Systems,
Constraints,
Transformation,
Interpretation

Language Workbenches

Typical Features

Goto Definition/Find Usages

Language Workbenches

Typical Features

- Goto Definition/Find Usages
- Error Markup/Quick Fixes

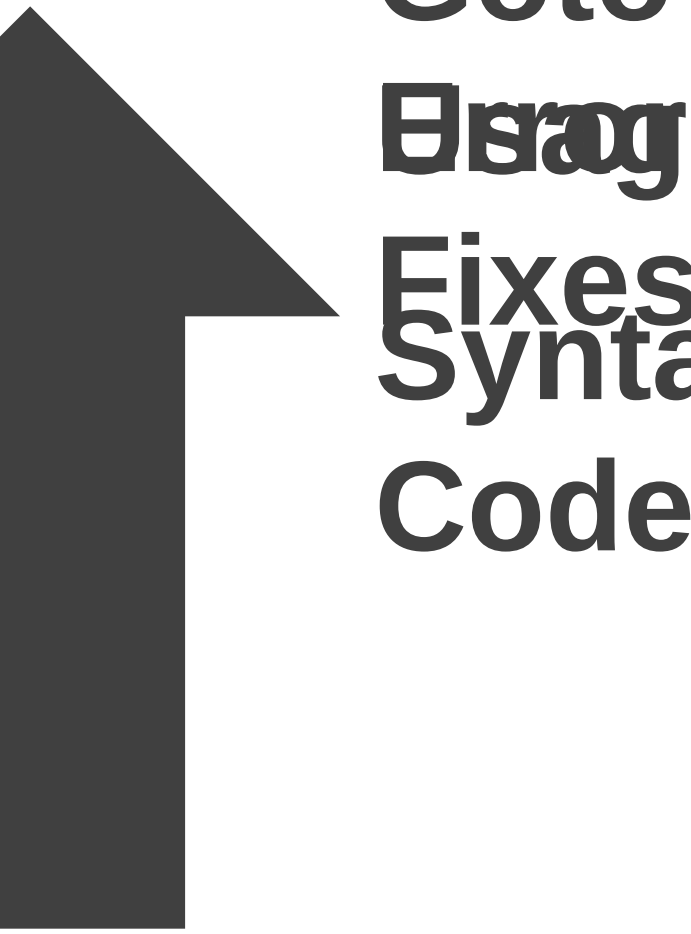
Language Workbenches

Typical Features

- Goto Definition/Find Usages
- Error Markup/Quick Fixes
- Syntax Highlighting

Language Workbenches

Typical Features



Goto Definition/Find Usages

Error Markup/Quick Fixes

Syntax Highlighting

Code Completion

Language Workbenches

Typical Features

- Goto Definition/Find Usages
- Error Markup/Quick Fixes
- Syntax Highlighting
- Code Completion
- Search/Replace

Language Workbenches

Typical Features

Goto Definition/Find Usages

Error Markup/Quick Fixes

Syntax Highlighting

Code Completion

Search/Replace

Refactoring

Language Workbenches

Typical Features

Goto Definition/Find Usages

Error Markup/Quick Fixes

Syntax Highlighting

Code Completion

Search/Replace

Refactoring

Debugging

Language Workbenches

Typical Features

Goto Definition/Find Usages

Error Markup/Quick Fixes

Syntax Highlighting

Code Completion

Search/Replace

Reporting

Refactoring

Debugging

Language Workbenches

Typical Features

Goto Definition/Find Usages

Error Markup/Quick Fixes

Syntax Highlighting

Code Completion

Search/Replace

Reporting

Refactoring

Visualization

Debugging

Language Workbenches

Typical Features

Goto Definition/Find Usages

Error Markup/Quick Fixes

Syntax Highlighting

Code Completion

Search/Replace

Reporting

Refactoring

Visualization

Debugging

Version Control

Language Workbenches

Typical Features

for any Language!

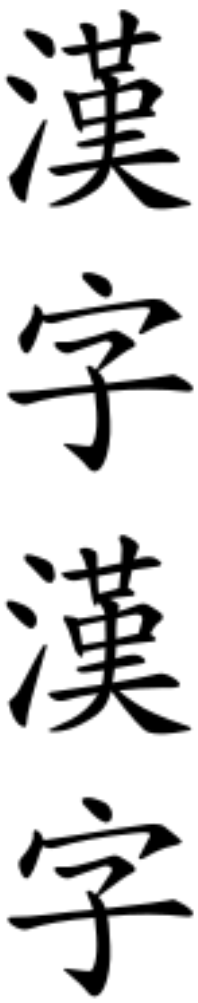
Language Workbenches

Typical Features

Language Workbenches are IDEs for arbitrary languages.

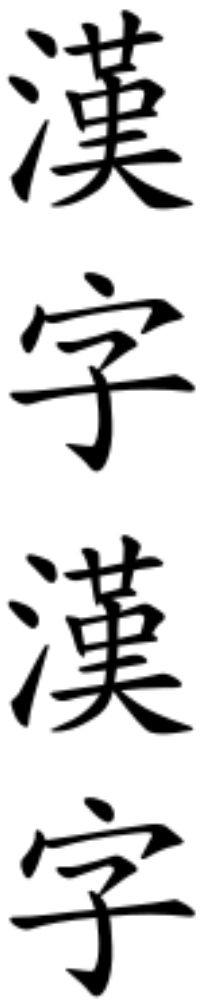
Languages / Language Extensions

Contribute Customizations



Languages / Language Extensions

Contribute Customizations



Language Definition,
Reuse, Extension,
Composition
Mixing Notations
Type Systems,
Constraints,
Transformation,
Interpretation

Languages / Language Extensions

Contribute Customizations

Goto Definition/Find Usages

Error Markup/Quick Fixes

Syntax Highlighting

Code Completion

Search/Replace

Reporting

Refactoring

Visualization

Debugging

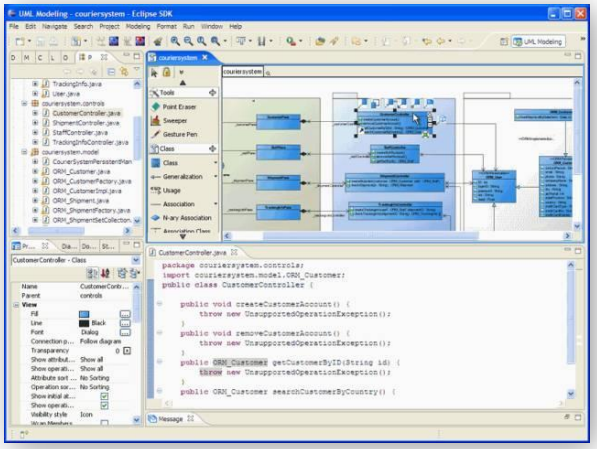
Version Control

Languages / Language Extensions

Additional Stuff

漢字漢字

Buttons Views
Menus Actions



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 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.

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).

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)!

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

Generic Tools, Specific Languages

Adaptability is built-in!

Extensions are first-class

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

An Example

An Example System

Language Engineering Embedded Software

Specific Languages

An Example System

Language Engineering Embedded Software

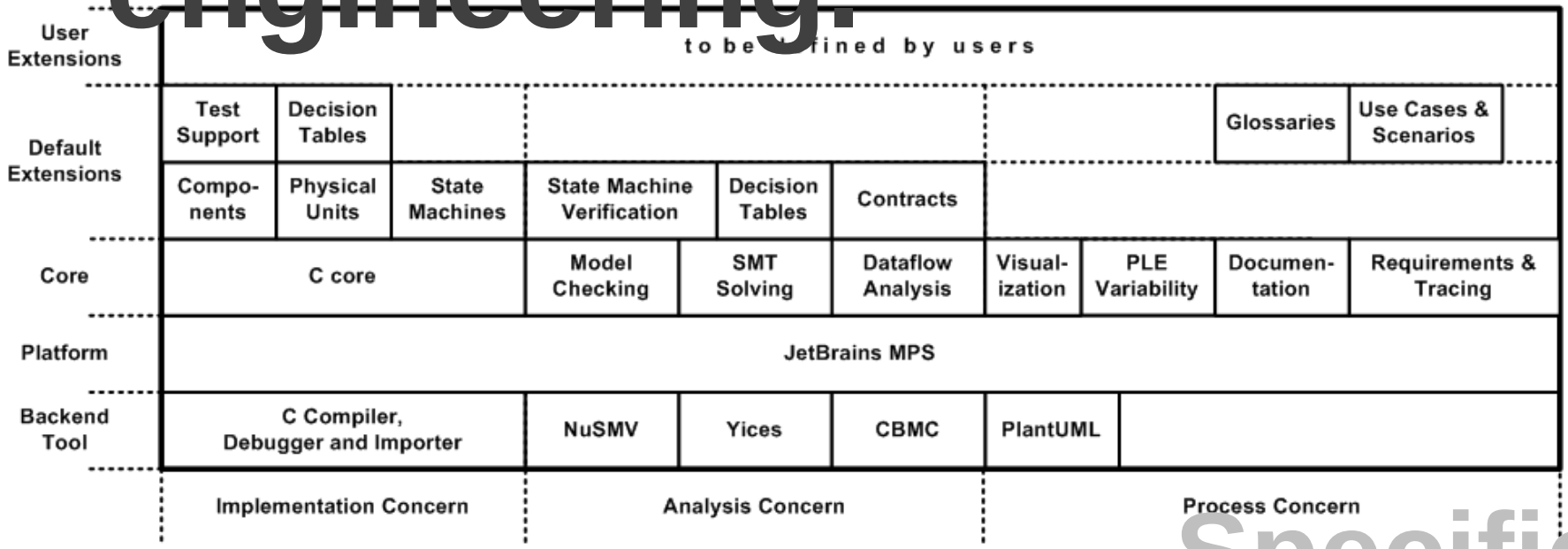
A collection of integrated languages for embedded software engineering.

Specific Languages

An Example System

Language Engineering Embedded Software

A collection of integrated languages for embedded software engineering.

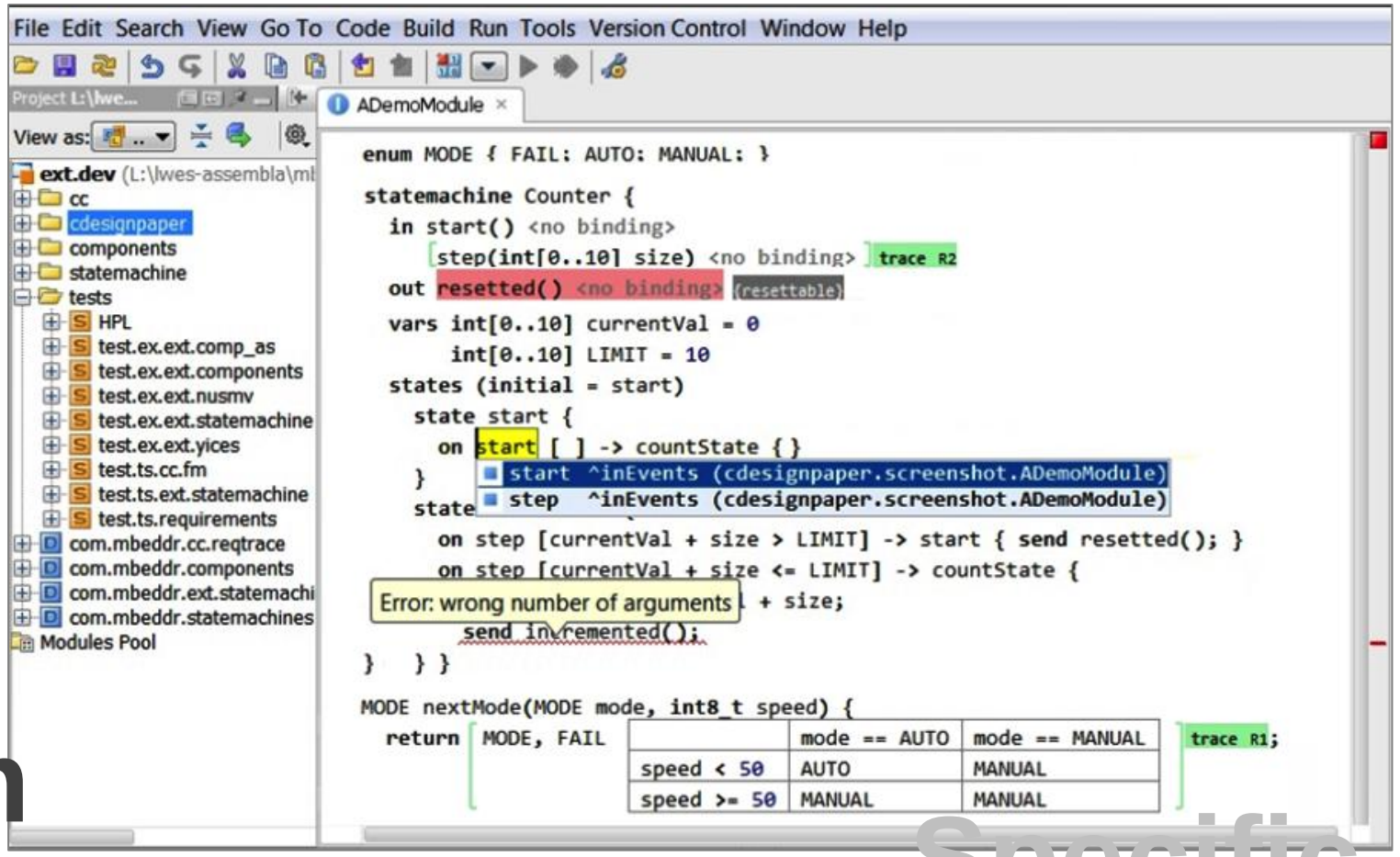


Specific Languages

An Example System

Language Engineering Embedded Software

An IDE for all Of them



Specific Languages

An Example System

Language Engineering Embedded Software

Open Source
Eclipse Public License

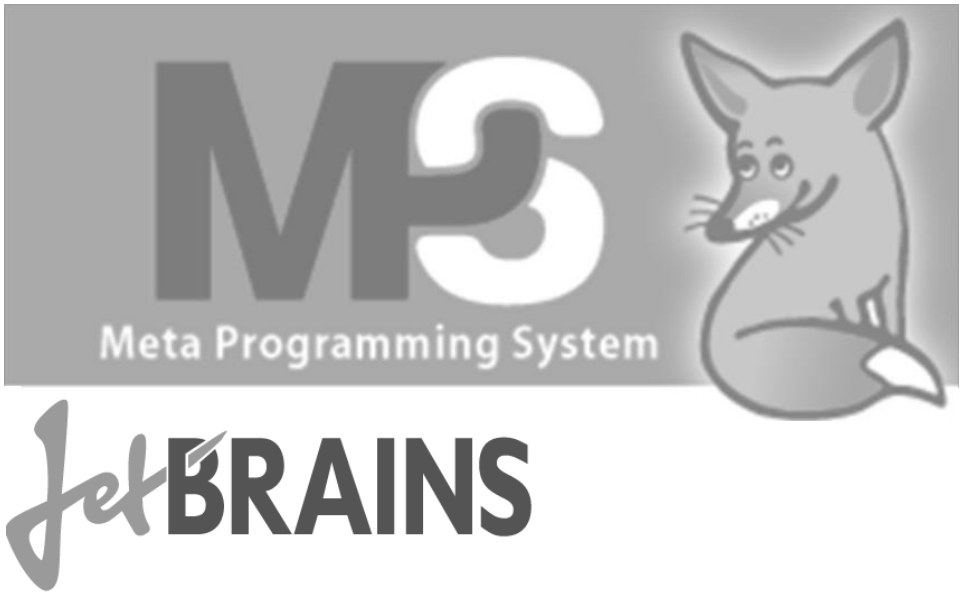
http://mbeddr.com

itemis fortiss

Specific Languages

An Example System

Built on JetBrains MPS



Generic Tool

An Example System

Built on JetBrains MPS

Projectional Editing

Textual/Symbolic/Tabular/(soon Graphical)

Multiple projections for the same language (in 3.0, due soon)

Modular language development, extension and embedding

An Example System

Built on JetBrains MPS

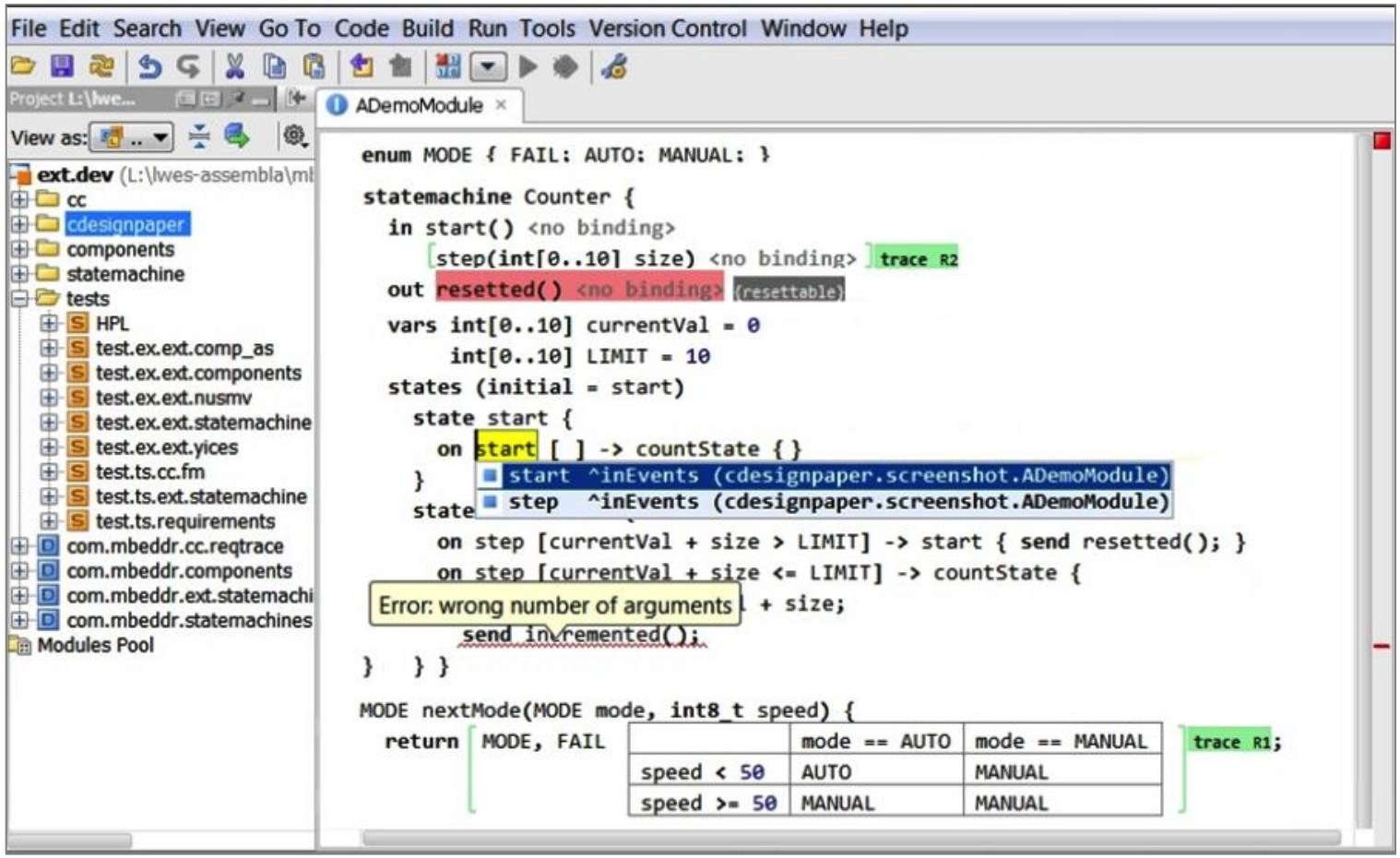
Support for language aspects such as type system, scopes, code completion, find usages, dataflow

Template-based approach for transformation and code generation with IDE support for target language in templates

Support for building extensible debuggers

An Example System

Built on JetBrains MPS



Generic Tool

An Example System

Built on JetBrains MPS

Open Source
Apache 2.0

http://jetbrains.com/mps

D E M O

Summing up

Summing Up

Key Points

To build meaningful tools, the data must be extended.

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

Summing Up

Key Points

Structured Data can be expressed as languages.

Languages are data plus syntax

Summing Up

Key Points

Language Engineering supports extension and composition

This supports adapting tools for specific domains

Summing Up

Key Points

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

We've got a lot of experience from regular

Summing Up

Key Points

Language Workbenches are the key enabling technology.

MPS is IMHO the most ful but it‘s not the power

Summing Up

Key Points

Let's build new classes of tools!

... which make extensibility a reality meaningful

The End.

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

The End.