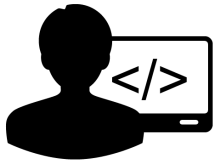
Convecton

A scalable language-oriented modeling and execution platform for the Web

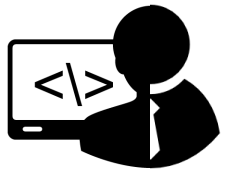
The Convecton Team @ itemis

Where we are and where we'll go

CONVECTON
capture•validate•execute



Sw Developer
Sw Architect



Business Analyst
Systems Engineer

Xtext

MPS

CONVECTON

Textual
Graphical
Files + VCS
Diff/Merge
IDE

Multi-Notation
Text Tables Math Graphical Prose

Files + VCS
Diff/Merge
IDE

Multi-Notation
Text Tables Math Graphical Prose Forms

Repository
Realtime-Collab
Browser/Cloud
Execution/Live

Convecton

High-Level Requirements and Decisions

Languages as in MPS

CONVECTON
capture•validate•execute

Reuse, Extension and Composition
of different languages with various levels of formality

Multiple mixed Notations
text, math, forms, tables, rich-text, graphical, custom.

Notation-specific interaction paradigms
Efficient typing for text, palettes for diagrams, etc.

Live like Excel

CONVECTON
capture•validate•execute

Program execution directly designed into the system

Support for incremental interpreters

Reactive Execution Architecture

Similar to Excel‘s incremental update of cells

Editor architecture optimized for feedback

Better way of annotating models with runtime values

Google-Doc Style Collaboration

CONVECTON
capture•validate•execute

Based on a shared repository

No files, no version control system

Realtime Collaboration

Based on OT, CRDT, etc.

Locking/Branching eventually supported

To provide a degree of isolation for users

Good Web Citizen

CONVECTON
capture•validate•execute

No IDE chrome around the notation
Radical reduction of accidental complexity

Models are REST resources
Facilitates integration with other tools

Notations rely on CSS to the degree possible
Supports designer-based styling

Deployable into the Cloud or on premises
Based on common abstractions such as Cloud Foundry

(Parts are) Embeddable in other web apps
For example, an embedded text editor with JSON IO

IT Integratable

CONVECTON
capture•validate•execute

Infrastructure integration through frameworks

LDAP, Authorization, etc.

Referencing of external data (via URLs)

Dedicated Support in the Meta Languages

Extensible Service Architecture

To support domain-specific model processing

Scalabe Like Amazon

CONVECTON
capture•validate•execute

By deploying to mainstream cloud infrastructures

... but not directly coding against their APIs

Design for scalability

For example, by relying on messaging middleware

Client/Server Distribution

Run local analyses locally, merge with „global“ results from the server-side parts of the overall model

State of the Art Meta Languages

CONVECTON
capture•validate•execute

Language Definition in MPS

No bootstrapping initially b/c of different audiences

Declarative/Functional Language Definition DSLs

For efficient and analyzable languages

Test-driven language development

Test every aspect (except editor) in the IDE

Components

CONVECTON
capture•validate•execute

Convecton consists of

a client	to render notations and interact with end users in the browser
a server	to store and process models, and to coordinate concurrent edits
an IDE	to allow language designers to define and test languages

Current State

the client	A demonstrator – see video.
the server	Nothing yet.
the IDE	Version 0.1 – see video.

Convecton

The Client

The Demonstrator

CONVECTON
capture•validate•execute

CONVECTON

- Coffee Maker
  - Functional Requirements
    - Coffee resources
      - Supplying water
      - Supplying coffee beans
      - Supplying milk
    - Coffee maker operation
      - Starting up the appliance
      - Make coffee
        - Prepare plain coffee
        - Prepare milk coffee
        - Prepare latte macchiato
  - Safety aspects of the coffee maker
    - Enter safe state when cover is opened

FR1_3 Supplying milk

A suction hose can be attached to the coffee maker. If attached it takes milk from an external milk jar.

Related to: no related requirements

Is Done? ☐
Priority Medium
Due date 25.02.2018

FR2 Coffee maker operation

The coffee maker has four main operational modes. These are ***StartingUp***, ***Ready***, ***Maintenance***, and ***StandBy***.

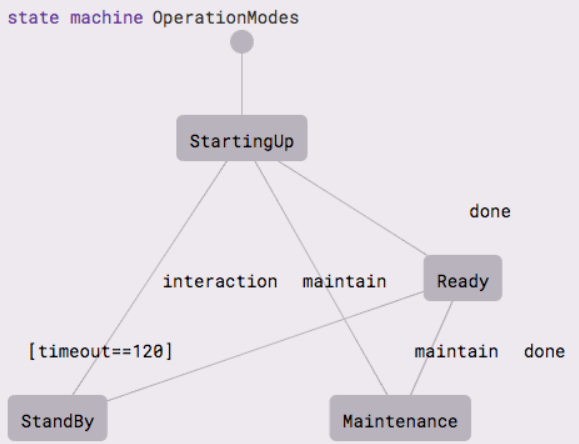
The coffee maker is in operation as soon as the main powerswitch is switched on by the user. If the user switches of the power switch or unplugs the power supply then the machine will go out of operation immediately without changing its mechanical state.

Related to: FR1

Is Done? ☐
Priority High
Due date 05.10.2017

```
state machine OperationModes {

  event done
  event maintain
  event interaction
  number timeout = 0

  initial is StartingUp

  The initial state of the coffee maker is  @StartingUp .
  It will be entered as soon as the user physically
  powers up the machine.

  state StartingUp {
    on done [] -> Ready
    on maintain [] -> Maintenance
  }

  state Ready {
    on   [ timeout == 120 ] -> StandBy
    on maintain [] -> Maintenance
  }

  state Maintenance {
    on done [] -> Ready
  }

  state StandBy {
    on interaction [] -> StartingUp
  }
}
```

▼ Graphical View



Client Architecture

CONVECTON
capture•validate•execute

Maintains a part of the overall model locally

for editing and rendering

Synchronizes changes with the server

which maintains the master copy of the model and resolves conflicts

Looks and feels like a web-app, but is „language-y“

Has forms, but also has textual and graphical editors

Maintains its own Virtual DOM

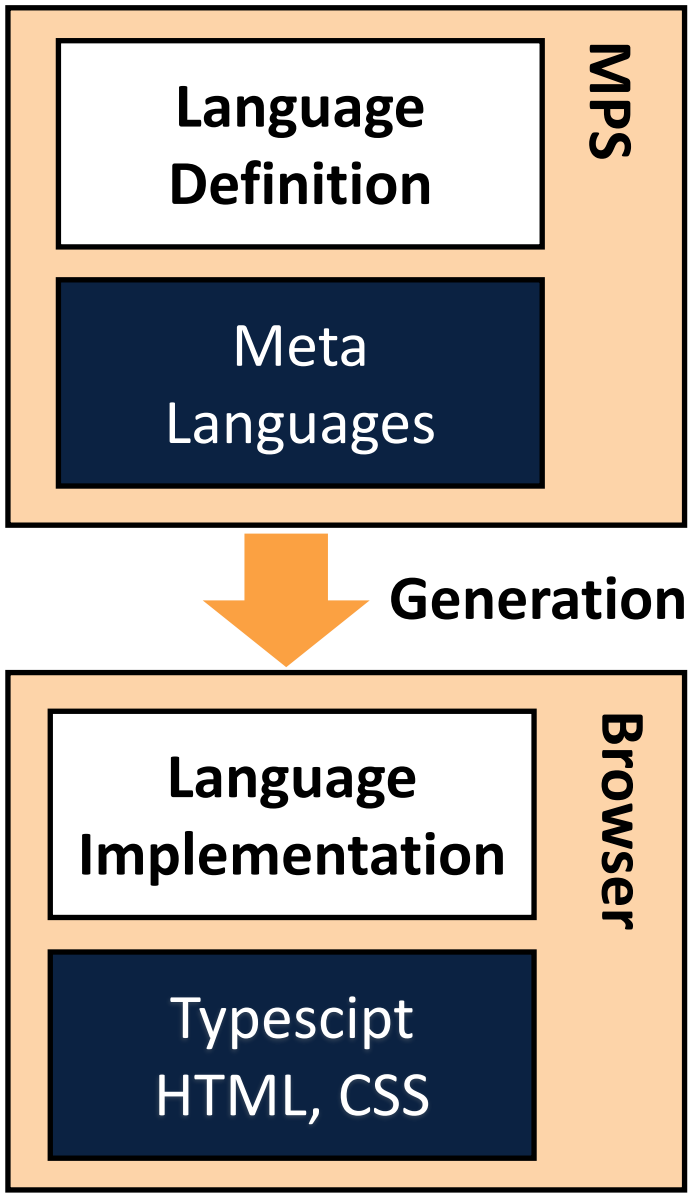
Which we call the cell tree

Convecton

The IDE

The IDE and the Runtime

CONVECTON
capture•validate•execute



Language Definition is done in Jetbrains MPS

From the definitions, we generate the language runtime, which is primarily based on Typescript

There is no dependency from the runtime to MPS. It‘s a regular WebApp.

Design Principles

CONVECTON
capture•validate•execute

Multiple meta languages, one per language aspect

structure, editor, type system, constraints, scopes.

Funclerative: basically functional, plus declarative „shortcuts“ where it makes sense.

All aspect languages embed common functional language, share elements and use the same „style“.

For example, all rely on meta functions and concept-based dispatch

Language Versioning and Testing built in from the start

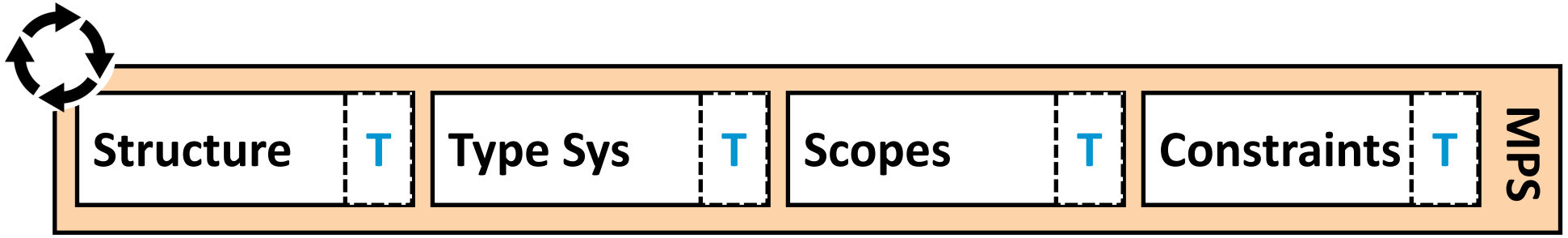
Tests can be executed in the IDE, without generation or a browser

No bootstrapping for now

Convecton itself targets non-developers, this does not fit with the needs of language developers.

Development Process

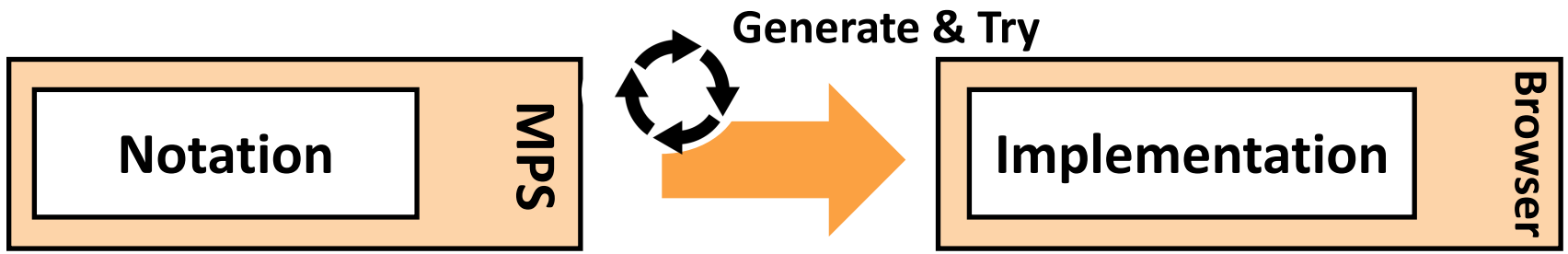
CONVECTON
capture•validate•execute



1

Define all non-visual language aspects, write tests, and run them in the IDE

Mimimal turnaround and accidental complexity



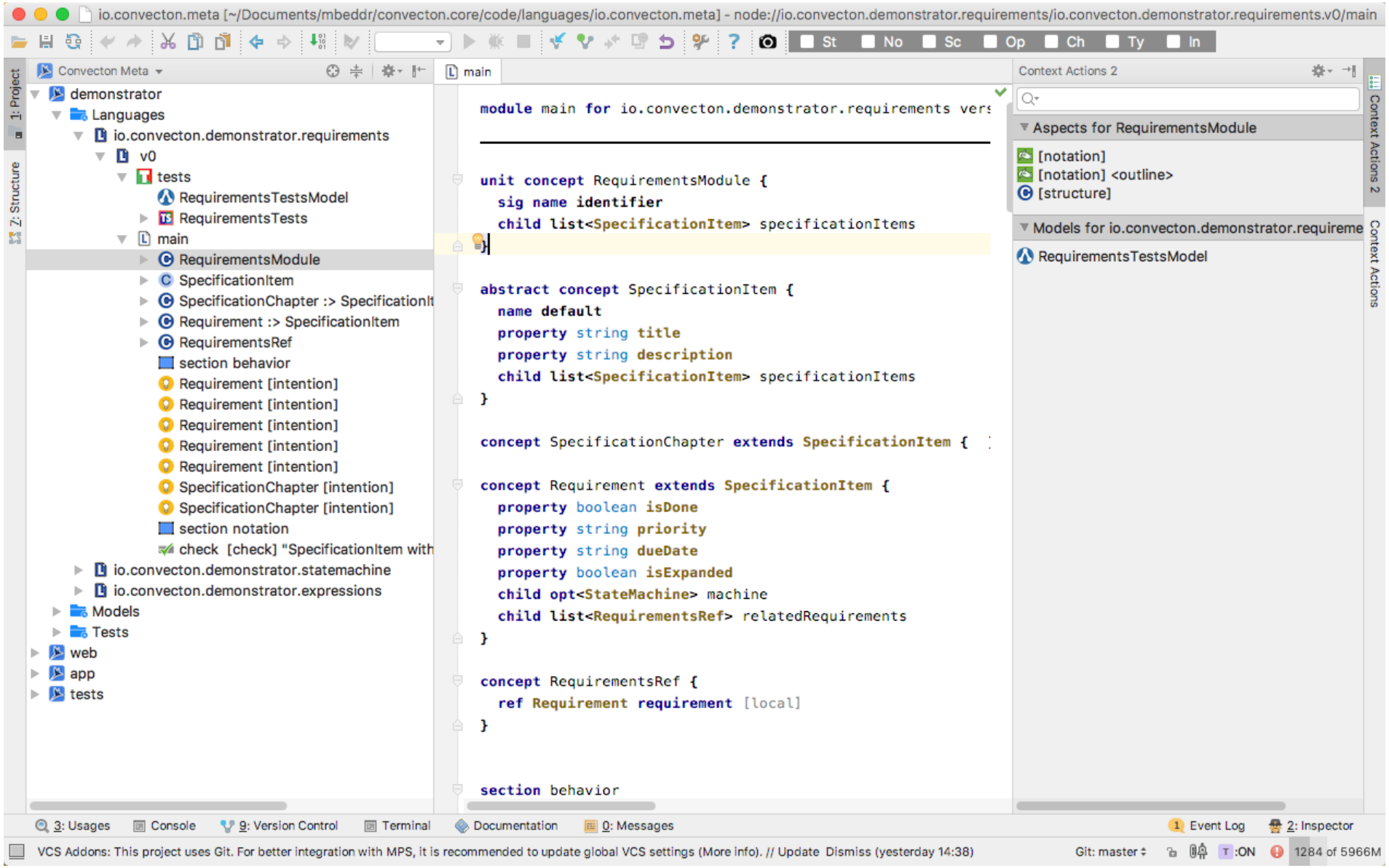
2

Define notation, generate implementation and run in local server to render notation

Increased, but acceptable turnaround time based on dynamic recompilation of the language implementation using webpack

THE IDE

CONVECTON
capture•validate•execute



STRUCTURE

CONVECTON
capture•validate•execute

```
unit concept RequirementsModule {
  sig name identifier
  child list<SpecificationItem> specificationItems
}

abstract concept SpecificationItem {
  name default
  property string title
  property string description
  child list<SpecificationItem> specificationItems
}

concept SpecificationChapter extends SpecificationItem {  }

concept Requirement extends SpecificationItem {
  property boolean isDone
  property string priority
  property string dueDate
  property boolean isExpanded
  child opt<StateMachine> machine
  child list<RequirementsRef> relatedRequirements
}

concept RequirementsRef {
  ref Requirement requirement [local]
}
```

NOTATION I

CONVECTON
capture•validate•execute

```
notation Requirement: grid row col[col-md-1] name
                          col[col-md-10] [property(title) anchorLink Requirement.name]
                      row col[col-md-12] [property(description)]
                      row col[col-md-1] "Is Done?"
                          col[col-md-2] checkbox(isDone)
                      row col[col-md-1] "Priority"
                          col[col-md-2] combo(priority)
                      row col[col-md-1] "Due date"
                          col[col-md-2] datepicker(dueDate)
                      row col[col-md-1] "Related"
                          col[col-md-5] children(relatedRequirements)
                      row col[col-md-6] child(machine)
                          col[col-md-6] hint(+graphical) -> [child(machine)]
                      row col[col-md-12] children(specificationItems)

notation RequirementsRef: tooltip on ref(requirement -> name)
                              tip [query(this.requirement)]

notation<outline> RequirementsModule: tree-container [children(specificationItems)]

notation<outline> SpecificationChapter: tree label title
                                            link SpecificationChapter.name
                                            component mvpViewer
                                            children specificationItems
```

NOTATION II

CONVECTON
capture•validate•execute

```
notation StateMachine: ["state machine" name "{" [children(nodes)] "}"]

notation StatemachineEvent: ["event" name]
notation StatemachineEventRef: ref(event -> name)

notation StatemachineVar: [child(type) name ["=" child(initial)]]
notation StatemachineVarRef: ref(var -> name)

notation IEmptyLine: "<no text>"

notation Documentation: [children(content)]
notation StringDocumentationContent: property(value)
notation StateRefContent: ["@" ref(referencedState -> name)]
notation NewLineContent: "<no text>"

notation Entry: ["initial is" ref(initial -> name)]
notation State: ["state" name "{" children(reactions) "}"]
notation Exit: ["exit" name]
notation FinalState: ["final" name]

notation Transition: [child(triggers) "->" tooltip on ref(target -> name) ]
                                                tip [query(this.target)]
notation TriggerSpec: ["on" children(triggers) child(guard)]
notation GuardSpec: ["[" child(guard) "]"]

notation<unguarded> Transition: [">" ref(target -> name)]
```

SCOPES + CHECKS

CONVECTON
capture•validate•execute

```
scope Transition::target -> navigate {
  pick from   StateMachine::nodes
       path   (node, parent) = parent.ancestors<StateMachine>
       filter (node, parent, candidate) = candidate.isInstanceOf<IState>
}


scope StatemachineVarRef::var -> navigate {
  pick from   StateMachine::nodes
       path   (node, parent) = node.ancestors<StateMachine>
}


check error on State
   condition (node) = node.ancestor<StateMachine>.nodes.ofConcept<State>.
                      where(it.name == node.name).size > 1
     message (node) = "State with same name already exists!"
```

INTENTIONS

CONVECTON
capture•validate•execute

```
intention for Requirement
  text (node) = "Add new Requirement"
  is applicable (node) = node.isDirectlyUnder<RequirementsModule>
  execute (node)/M = node.addRequirementGlobal()

intention for Requirement
  text (node) = "Add new Requirement"
  is applicable (node) = isSome(node.ancestor<Requirement>)
  execute (node)/M = node.addRequirement()

intention for Requirement
  text (node) = "Add new sub requirement"
  is applicable (node) = node.ancestor<RequirementsModule>.specificationItems.
                         specificationItems.size < 10
  execute (node)/M = node.addRequirementAsChild()
```

Convecton

Challenges

Challenges

CONVECTON
capture•validate•execute

Incrementality

In particular, for model-to-model transformations

Collaboration

Through OT or CRDTs

Immutability vs. Performance

It‘s not quite as painfree as we thought, even with the „recommended“ libs