Reporting

Component quality/overview

Grammar overview

Type	Valid	Parse	Resolve	Capitalized	UniqueNames
EMAM	✗	✓	✗	—	—
EMAM	✗	✓	✗	✓	✓
EMAM	✓	✓	✓	✓	✓
EMA	✗	✓	✗	✓	✓
EMAM	✓	✓	✓	✓	✓
EMAM	✓	✓	✓	✓	✓
EMAM	✓	✓	✓	✓	✓
EMAM	✗	✓	✓	✓	✓
EMAM	✗	✓	✓	✓	✓
EMAM	✗	✓	✓	✓	✓
EMAM	✓	✓	✓	✓	✓

Malte Heithoff
Praktikum
am Lehrstuhl für Software Engineering
RWTH Aachen

Reporting

- Creates a report for each component in EmbeddedMontiArc, MontiCore and MontiSim
- Checks all CoCos for each components (e.g. component names start with a capital letter)
- Order everything after the root package
- Create the visualisation for each component (see visualisation from Manuel Schrick)
- Log everything
- Print collected data in a json file
- Push data and visualisation files to gh-pages
- Display the report in a proper way (table with specialized features)
- Three parts:
  - Java part (information generation)
  - Travis / shell script part (publish to gh-pages)
  - HTML / JS part (Report Representation)

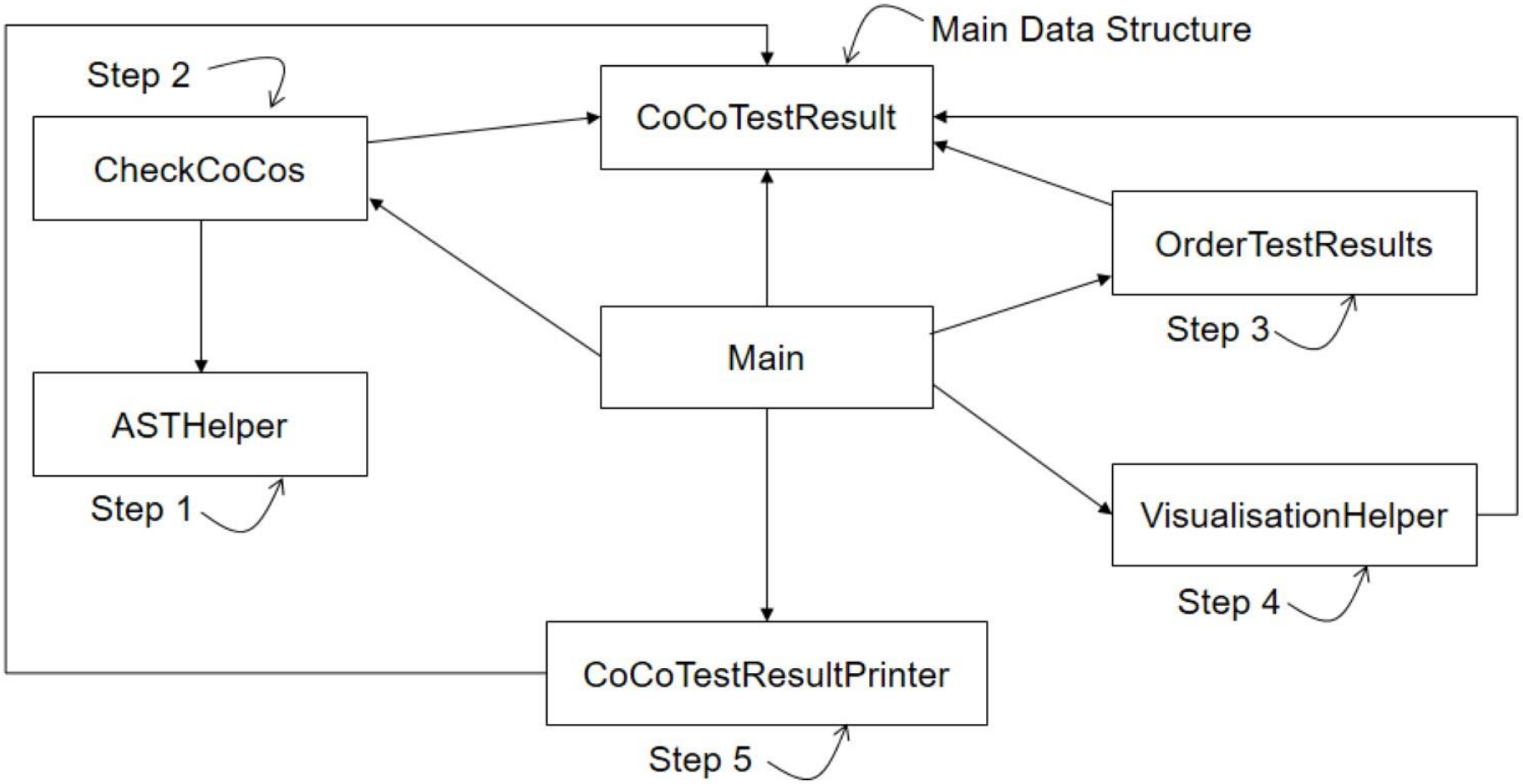
Outline

1. Information Generation
2. Report Representation
3. Publishing
4. Code Quality
5. Future Work

Outline

1. Information Generation
2. Report Representation
3. Publishing
4. Code Quality
5. Future Work

Reporting – Structure Data Generation



Classes – Data Generation 1

- Main
  - Reads user input and manages the control flow
- CoCoTestResult
  - Main data structure to store every piece of information for a model (e.g. CoCoTest results or whether it is resolvable)
- ASTHelper
  - Parses and resolves (if possible) all given models
- CheckCoCos
  - Checks all CoCos for all models given and extract the results
- OrderTestResults
  - Order each CoCoTestResult after its root package and subcomponents

Classes – Data Generation 2

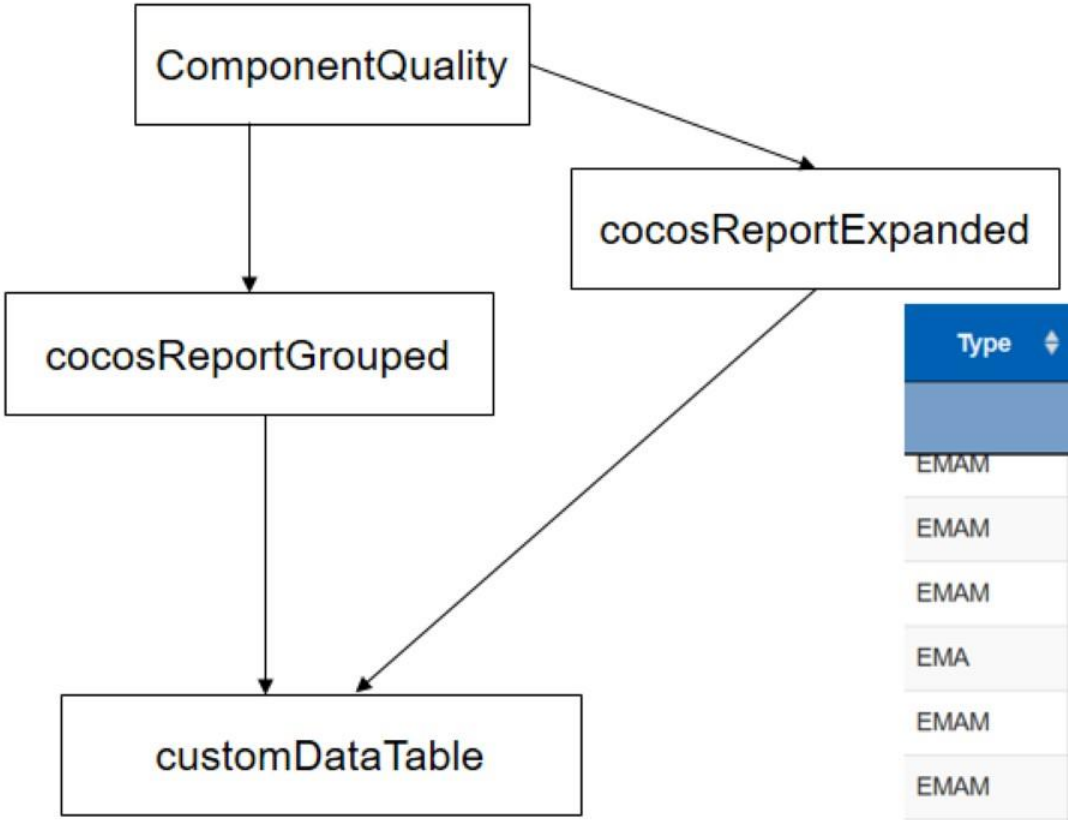
- VisualisationHelper
  - Call the Visualisation Jar (Manuel Schrick) for every root model (multithreaded)
  - Pass down the visualisation information to all sub-components
- CoCoTestResultPrinter
  - Prints a json file with all test information + GitHub links + visualisation links + onlineIDE links (Jean-Marc)

Outline

1. Information Generation
2. Report Representation
3. Publishing
4. Code Quality
5. Future Work

Malte Heithoff
Software Engineering
RWTH Aachen

Reporting – Structure Data Generation



Type	Valid	Parse	Resolve	Capitalized	UniqueNames
EMAM	✗	✓	✗	—	—
EMAM	✗	✓	✗	✓	✓
EMAM	✓	✓	✓	✓	✓
EMA	✗	✓	✗	✓	✓
EMAM	✓	✓	✓	✓	✓
EMAM	✓	✓	✓	✓	✓
EMAM	✓	✓	✓	✓	✓
EMAM	✗	✓	✓	✓	✓
EMAM	✗	✓	✓	✓	✓
EMAM	✗	✓	✓	✓	✓
EMAM	✓	✓	✓	✓	✓

Report Representation

- Display report on gh-pages as a table
  - JQuery – Datatable is used as base JavaScript library
  - There are two tables, but only one visible at a time (managed in componentQuality.html/js)
    - 1. with every component grouped by its root package
    - 2. with every component itself (expanded)

1.

Expand

EmbeddedMontiArc (1322)	valid: 911
	demonstrator.simulator1 (4)
	simulator1.ConstantVelocity
	simulator1.GameOverTrigger
	simulator1.MainController
	simulator1.SteeringControl
	Documentation.controller (1)
	Documentation.controller (5)

2.

Group

EmbeddedMontiArc (1322)	valid: 911	invalid: 411
	EMAM2RosCpp/ba.intersection.ConflictComput	
	EMAM2RosCpp/ba.intersection.ConflictToStopC	
	EMAM2RosCpp/ba.intersection.ConflictToStopL	
	EMAM2RosCpp/ba.intersection.DeltaTrajectory	
	EMAM2RosCpp/ba.intersection.TrajectoryToSto	
	EMAM2RosCpp/ba.util.RelToAbsTrajectory	
	EMAM2RosCpp/test.BasicConstantAssignment	

Report Representation

- For each test category there is an according column

Type	Valid	Parse	Resolve	Capitalized	UniqueNames	TypePar	EndQualif	ParOrder
EMAM	✗	✓	✓	✓	✓	✓	✓	✓
EMAM	✗	✓	✗	—	—	—	—	—
EMAM	✗	✓	✗	—	—	—	—	—
EMAM	✓	✓	✓	✓	✓	✓	✓	✓
EMAM	✓	✓	✓	✓	✓	✓	✓	✓
EMAM	✗	✓	✗	—	—	—	—	—

- Tick for passing the category (Valid is only passed if every other category is passed)
- Cross for not passing
- Minus for not getting to the corresponding category (e.g. parsing failed before testing whether it can be resolved)

Report Representation – Extra Features

- For better readability the following features were added. Features are accessable for other usage in customDataTable.js
  - Expansion of child components (see previous slide)
  - A floating header with the project information
  - Current project name on top of the window (grouping)

EmbeddedMontiArc (1322) **valid: 911** **invalid: 411**

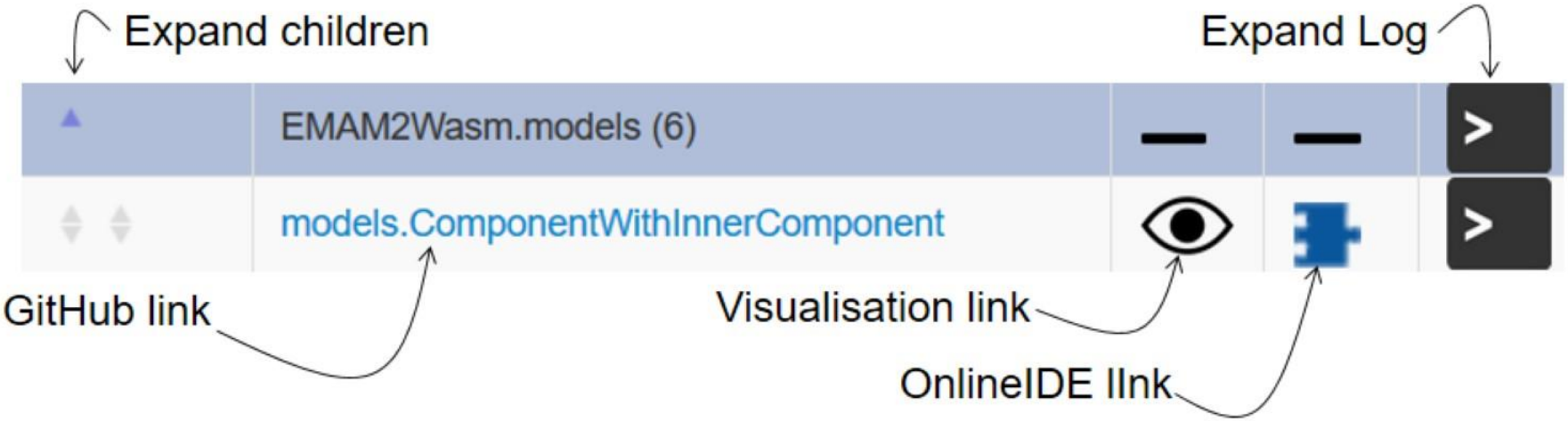
  - Log mechanic

EMAM2RosCpp/test.InstanceArrayComp

Path:	/EMAM2RosCpp/src/test/resources/test/InstanceArrayComp.emam
Log output:	[INFO] do Parser Test ========== [INFO] Parser Test success [INFO] do Resolve Test ========== [INFO] Resolve Test success [INFO] do CoCo Tests ========== [ERROR] 0xAC008 Port basicPorts1.in1 of subcomponent test.InstanceArrayComp.basicPorts1 is not used! [ERROR] 0xAC008 Port basicPorts1.in2 of subcomponent test.InstanceArrayComp.basicPorts1 is not used! [ERROR] 0xAC008 Port basicPorts1.out1 of subcomponent test.InstanceArrayComp.basicPorts1 is not used!

Report Representation - Controls

- The Main Controls are:
  - Switching between the grouped and expanded view of all components
  - Expand children in the grouped view
  - Link to the GitHub – file
  - Link to the Visualisation
  - Link to open with the onlineIDE
  - Open the Log



Outline

1. Information Generation
2. Report Representation
3. Publishing
4. Code Quality
5. Future Work

Travis Build

- Each night Travis builds and publishes the report
- Step 1: Pull all repositories from EmbeddedMontiArc, MontiCore and MontiSim
- Step 2: Create the report from all models inside
- Step 3: Zip all models (in order for the onlineIDE to work)
- Step 4: Push the report + visualisation files + zip files to gh-pages (for this, a special deploy key was created)
- The script files are accessable and reusable for other purposes in the EmbeddedMontiArc/reporting repository

Outline

1. Information Generation
2. Report Representation
3. Publishing
4. Code Quality
5. Future Work

Code Quality

- The tests were suspended due to the amount of changes in the format of the test results
- CoCo checks are only reused and should work properly
- There is a build each night, so the build is stable
- Code Segments are well sorted and refactored
- Code can be reused with little effort
  - ASTHelper
  - Orderable Test Results
  - Visualisation Generation
  - All JavaScript methods are hold generally

Outline

1. Information Generation
2. Report Representation
3. Publishing
4. Code Quality
5. Future Work

Future Work

- There already is a report for all grammars for MontiCore
  - Lists every Grammar, groups it after mainGrammars and others
  - Offers the GitHub link and onlineIDE view
- Creation of a web assembly with emam2wasm is currently in work
- In EmbeddedMontiArcStudio there is already a feature build in to automatically test all stream tests and present the results
- Web assembly generation and stream testing are heavily time intensive and therefore are not suitable for Travis (due to the time restriction for a build of one hour)