Experiences with an Internal DSL in the IoT Domain

Matthias Tichy, Jakob Pietron, David Mödinger, Katharina Juhnke, Franz Hauck

Software Engineering
Programming Languages

SORRIR

ulm university universität ulm

Motivation

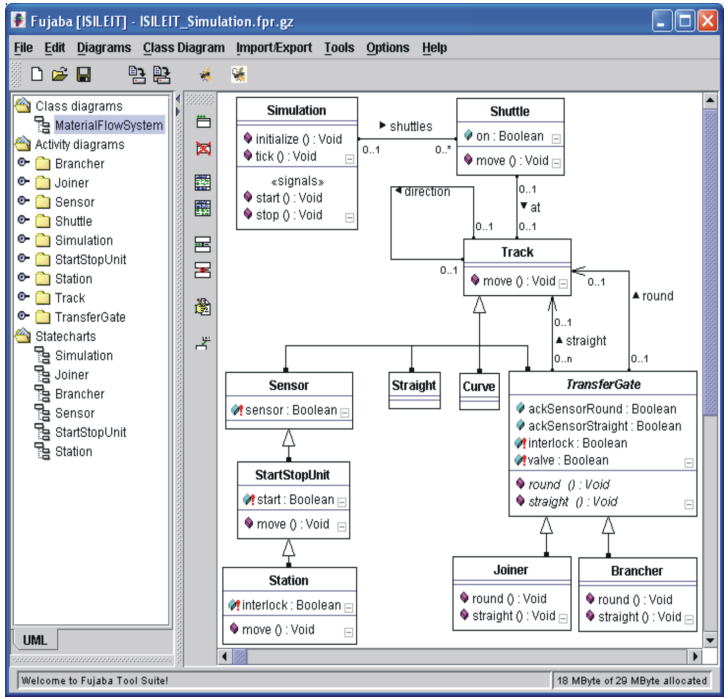
Context

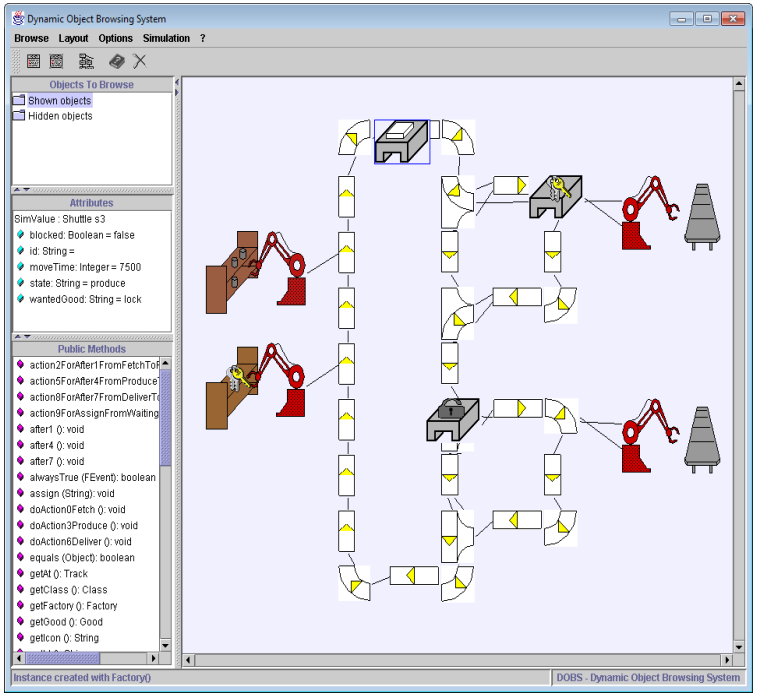
- Development of resilient IoT systems
- Support for automatic orchestration, active replication, degradation

Use Case

- Smart parking garage

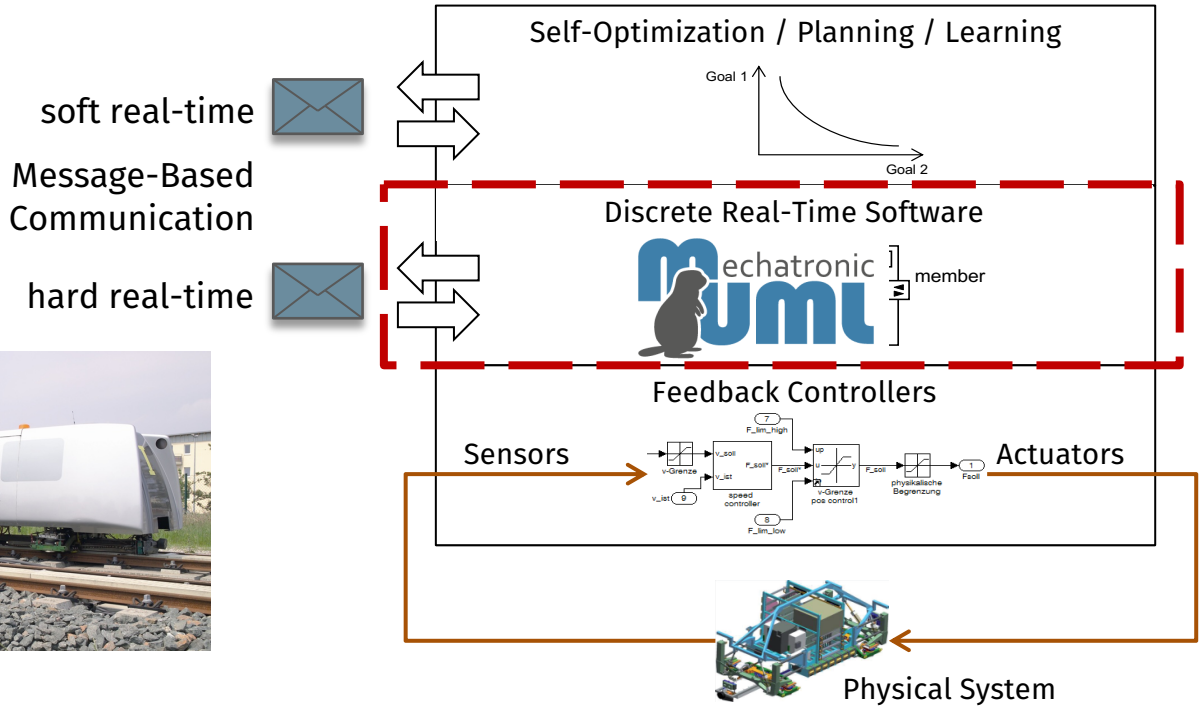
History: Fujaba (Own Framework & DSL, 1996-2005)





Sven Burmester, Holger Giese, Jörg Niere, Matthias Tichy, Jörg P. Wadsack, Robert Wagner, Lothar Wendehals, Albert Zündorf:
Tool integration at the meta-model level: the Fujaba approach. Int. J. Softw. Tools Technol. Transf. 6(3): 203-218 (2004)

History: MechatronicUML (Eclipse-DSL, 2005-2014)



Steffen Becker, Stefan Dziwok, Thomas Gewering, Christian Heinzemann, Uwe Pohlmann, Claudia Priesterjahn, Wilhelm Schäfer, Oliver Sudmann, and Matthias Tichy. MechatronicUML - Syntax and Semantics. Technical report, Software Engineering Group, Heinz Nixdorf Institute, University of Paderborn, 2014.

Related Work

Survey by Nguyen et al.

- Many DSLs in the domain
- Many approaches follow a component-based, data-flow-oriented paradigm

Graphical DSLs, e.g., Node-RED, DSL-4-IoT

Textual DSLs, e.g., ThingML, DoS-IL, SALT

- Extensive development effort required for complete tool chain

Our focus

- Resiliency (need to have control of state)
- Experienced developers and the existing programming language ecosystem
- But restricted by the ecosystem and the programming language

Goal of DSL Development

Explore the Sweet Spot of

- Supporting MDE and reaping its benefits
  - Without the huge effort to build a **usable** (Eclipse-based) External DSL
  - Without extensive training for users

- by tightly integrating with Programming Language Ecosystem:
  - Test-Frameworks, Logging-Frameworks
  - Libraries, Libraries, Libraries
  - IDEs and their features (Debugger, Recommenders, ...)

- Target Group: Experienced Software Developers

Our TypeScript Internal DSL: What do we support?

Component model

- Components. Ports
- 1:1, 1:n, m:1, n:m connections
- Architectural Configuration
- Mux-/DeMux for Active Repl.
- Integration with MQTT, REST

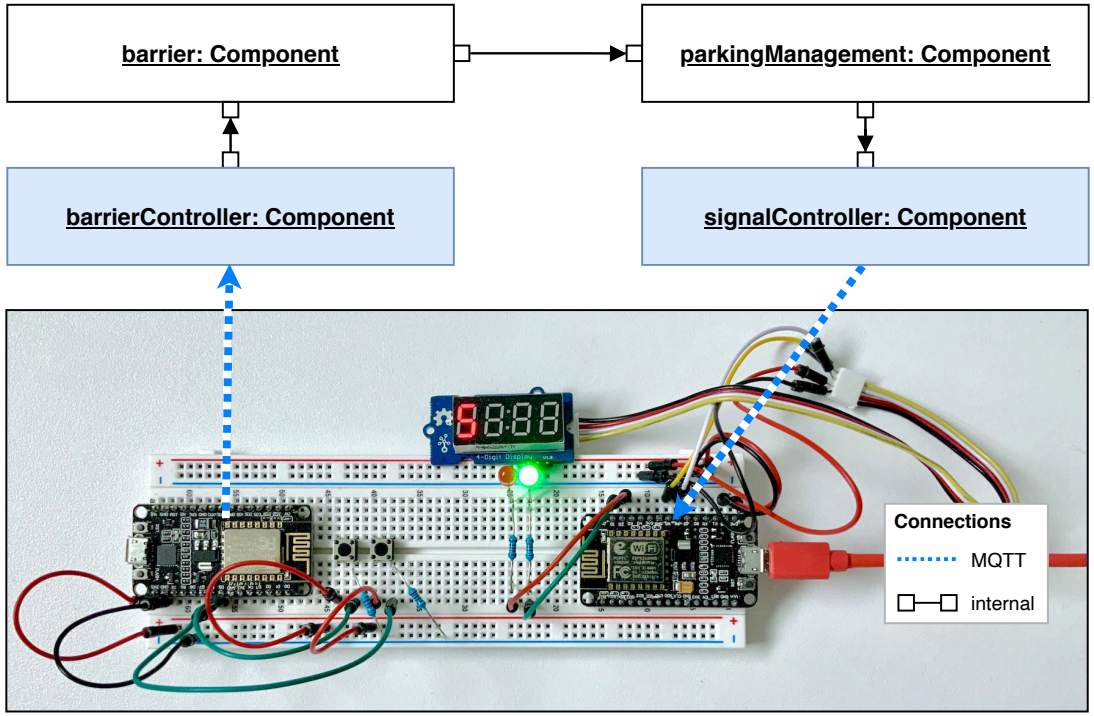
Behavior

- Interpreter
- Side-Effect Free / Pure
- State Space Exploration

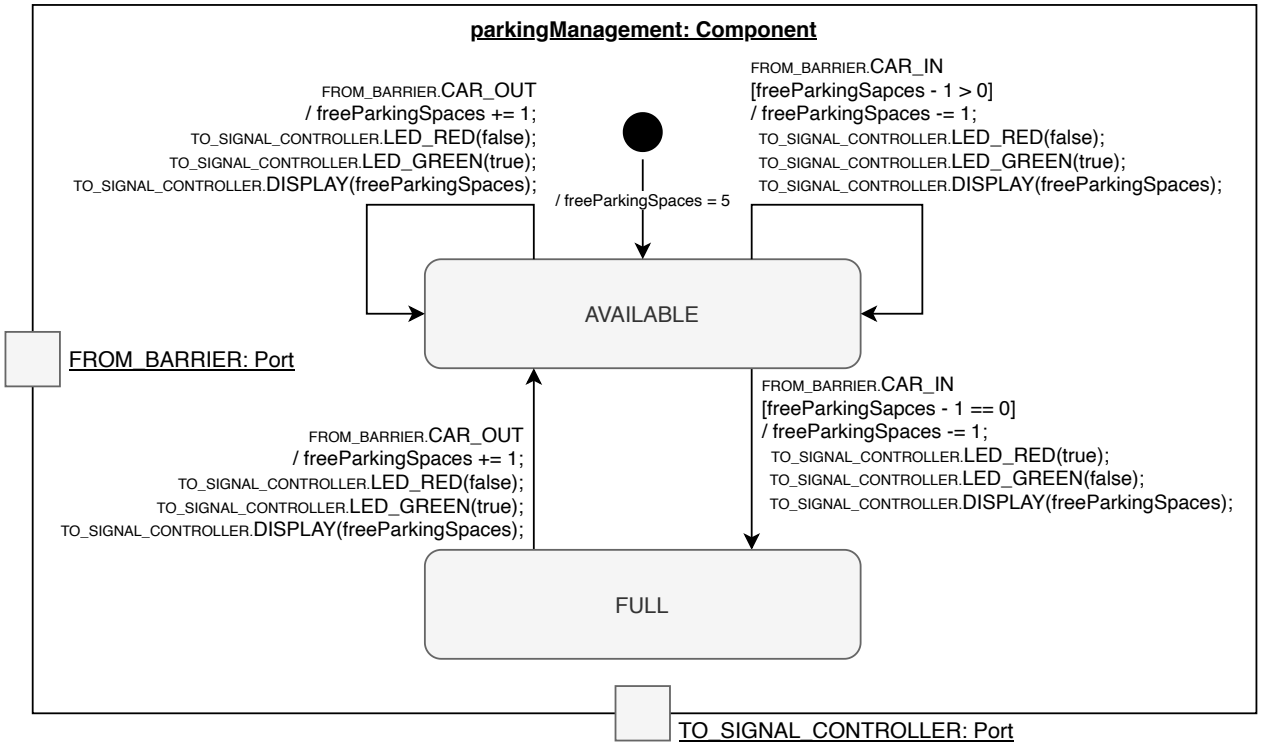
State machines

- Discrete States / Events
- Abstract State
- Events with parameters
- Internal communication
- External communication
- Actions, conditions via functions
- Type Safety via Generics
- Functional interface
- Visualization to DOT, TGF
- Web GUI via React

Running Example: Simplified Parking Garage



Running Example: Simplified Parking Garage



Our TypeScript Internal DSL: „Meta Model“ Excerpt

```
interface Component<F, M, E, P> {
  name: string,
  ports: Port<E, P>[],
  step: (current: State<F, M, E, P>) => State<F, M, E, P>,
  allSteps: (current: State<F, M, E, P>) => State<F, M, E, P>[],
}
interface Transition<F, M, E, P> {
  sourceState: F,
  event?: [E,P?],
  condition?: (myState: M, event?:Event<E, P>) => Boolean,
  action?: (myState: M, raiseEvent:RaiseEventCallBack<E, P>, event?: Event<E, P>) => M,
  targetState: F,
}
interface StateMachine<F, M, E, P> {
  transitions: Transition<F, M, E, P>[],
}
```

Our TypeScript Internal DSL: „Model“ Excerpt

```
enum ParkingManagementStates { AVAILABLE, FULL}
type ParkingManagementAbstractState = { readonly freeParkingSpaces : number;}
const sm: StateMachine<ParkingManagementStates, ParkingManagementAbstractState, EventTypes, PMPorts> = {
  transitions: [
    {
      sourceState: ParkingManagementStates.AVAILABLE,
      targetState: ParkingManagementStates.AVAILABLE,
      event: [EventTypes.CAR_IN, ParkingManagementPorts.FROM_BARRIER],
      condition: myState => myState.freeParkingSpaces - 1 > 0,
      action: (myState, raiseEvent) => {
        const newState = {… freeParkingSpaces: myState.freeParkingSpaces-1;
        raiseEvent(   {type: EventTypes.LED_RED, port: PMPorts.TO_SIGNAL_CONTROLLER, payload: {status: false}});
        raiseEvent(   {type: EventTypes.LED_GREEN, port: PMPorts. TO_SIGNAL_CONTROLLER, payload: {status: true}});
        raiseEvent(   {type: EventTypes.DISPLAY, port: PMPorts.TO_SIGNAL_CONTROLLER,
                      payload: {fs: newState. freeParkingSpaces}});
        return newState;
      }
    },
…
```

Our Experiences

- Development is super productive and super smooth (no battling with Eclipse and EMF peculiarities)
- Super quick turn-around from code change to see effect
- Programming language ecosystem super helpful
- TypeScript Compiler automatically does a lot of well-formedness checks, e.g., only connections can only be created between ports with compatible event definitions
- Side-effect free function design enable easy test-driven development and state space exploration
- Modern functional programming APIs are a good replacement for OCL

Conclusion and Future Work

- Our own experiences are fairly positive (maybe, we are at the 20%/80% spot)
- Framework is available via NPM
- Our colleagues from the distributed systems and security groups can read and change our code ☺

- Future Work
  - Monitoring of events based on state machines specification
  - Supporting Degradation in response to failures
  - Automatic Orchestration: Failure → Restart somewhere else
  - Application of Fault Tolerance Patterns to component structures

Our Code is Available Online

- **Framework**
  https://www.npmjs.com/package/sorrir-framework
- **Web Demo**
  https://sorrir.github.io/web-demo/
- **Hardware Testbed**
  https://github.com/sorrir/hardware-testbed