Modeling Non-Functional Application Domain Constraints for Component-Based Robotics Software Systems

DSLRob 2015

Alex Lotz\textsuperscript{1}, Arne Hamann\textsuperscript{2}, Ingo Lütkebohle\textsuperscript{2}, Dennis Stampfer\textsuperscript{1}, Matthias Lutz\textsuperscript{1} and Christian Schlegel\textsuperscript{1}

In bilateral cooperation between the
\textsuperscript{1}University of Applied Sciences Ulm and the
\textsuperscript{2}Robert Bosch GmbH
• MDSE applied for Robotics Component Models, e.g.:
  – RobotML (from PROTEUS)
  – OMG RTC (with OpenRTM-aist reference implementation)
  – BRICS Component Model (BCM)
  – SmartSoft (and the SmartSoft MDSD Toolchain)
  – etc.
• These approaches already facilitate the development of functional components by robotics experts
• But: what about non-functional application-related aspects such as responsiveness and deterministic behavior?
Navigation Example

BaseServer
- publish odometry and receive velocity commands
- push timed (10 Hz) pose

CDL
- send (sporadic) velocity (vx/vw)
- push timed (10 Hz) pose
- calculate speed to next goal-position using:
  - current laser scan
  - robot contour
  - kinematic and dynamic properties

Laser
- get laser-scan from HW, attach curr. pose and publish to all subscribers

Push newest (sporadic) laser-scan

Planner
- plan next goal to goal position using newest map

Push newest (sporadic) map

Mapper
- update map using current laser-scan

How often does the robot calculate a new plan? (thus reacting to major changes in the map)

How fast can a robot react to a new obstacle?
Component's internal view

- **Causality**: is there a certain „in-port“ triggering the execution of the „CDLTask“ and thus influencing the behavior of the „out-port“?

- **Latency**: how long does it take for a message token to traverse the internal parts of a component (buffers, update-rates, etc.)?

- **Separation of concerns**: who is actually responsible for designing such issues (i.e. component developer or system integrator)?
System integration problems before our modeling approach

- **Causality**: propagation rules for communication behavior in a chain of interconnected components are unknown on model level
- **Latency and Jitter**: Data aging from source to destination is massively influenced by individual component implementations
- **Separation of concerns**: system integrators need to investigate and modify components' internal implementations
- Ecore meta-model is separated into two packages; one for *component development* (left) and the other for *system configuration* (right)

- Each package can be realized by an individual DSL (either textually or graphically)
A Component consists of In-/Out-Ports and Tasks

Tasks can use data from several InPorts and can generate data for several OutPorts

*Optional* ActivationConstraints can define implementation-specific Task execution behavior

If no ActivationConstraints were defined, the Task execution behavior is purposefully left open for later system configuration

ActivationConstraints influence both:

- the required communication behavior for the related InPorts and
- the provided communication behavior for the related OutPorts
The main concern here is the definition of component-instances and their initial connections.

In addition, ActivationSources clearly specify the individual Task’s execution behavior (i.e. Periodic, Sporadic or DataTriggered).

Moreover, CauseEffectChains allow to define sensor-to-actuator couplings and assign individual end-to-end latency specifications.

The system configuration model is kept in sync with the referenced component model definitions.

The system configuration model generates consistent component configurations.
A graphical DSL for Component Definition

- Dashed lines between Tasks and In-/Out-Ports define the functional dependency on input data from several in-ports and/or responsibility to generate data for several out-ports
  - This already allows to fully implement the component without presuming application specific timing requirements
- ActivationConstraints can further specify implementation-specific restrictions (e.g. caused by involved HW drivers or algorithmic needs)
  - These constraints can be either a changeable boundary condition or a strict implementation constraint
A textual DSL for System Configuration

- System Integrator (i.e. the application domain expert) is responsible to instantiate components and to initially connect their ports (see bottom right).

- For each Task in a component-instance an individual ActivationSource is chosen such that potential ActivationConstraints are not violated.

- The relevant sensor-to-actuator couplings are defined by CauseEffectChains.
  - E2ELatencySpecs further allow to directly annotate application specific end-to-end latency specifications.
System Configuration Model - Interpretation and Evaluation

```plaintext
SysConfigModel NavigationScenario {
    CompInstance FrontLaser instantiates Laser {
        TaskRef MapperTask {
            trigger DataTriggered { inPortRef LaserClient prescale 10 }
        }
    }
    CompInstance Mapper instantiates Mapper {
        TaskRef MapperTask {
            trigger DataTriggered { inPortRef LaserClient prescale 4 }
        }
    }
    CompInstance Planner instantiates Planner {
        TaskRef OATask {
            trigger PeriodicTimer 10.0 Hz
        }
    }
    CompInstance Base instantiates Base {
        TaskRef VelocityCommandTask {
            trigger DataTriggered { inPortRef VelocityClient }
        }
        TaskRef PoseUpdateTask {
            trigger Sporadic { maxActFreq 30.0 Hz minActFreq 30.0 Hz }
        }
    }
    CauseEffectChain FastReactiveNavigationLoop {
    }
    endCompRef Base
    E2ELatencySpecs { maxAge 0.1 sec maxReaction 0.1 sec }

    CauseEffectChain PlannedNavigationLoop {
    }
    endCompRef Base
    E2ELatencySpecs { maxAge 1.0 sec maxReaction 1.0 sec }
}

// connections related to the fast-reactive navigation loop
Connection { client FrontLaser.BaseStateClient server Base.BaseStateServer }
Connection { client ObstacleAvoidance.LaserClient server FrontLaser.LaserScanServer }
Connection { client Base.VelocityClient server ObstacleAvoidance.VelocityServer }
// connections related to the slower, planned navigation loop
Connection { client Planner.MapClient server Mapper.MapServer }
Connection { client ObstacleAvoidance.MapClient server Planner.GoalServer, ObstacleAvoidance.VelocityServer }
Connection { client Planner.BaseStateClient server Base.BaseStateServer }

CauseEffectChain FastReactiveNavigationLoop {
    ServiceRef Base.BaseStateServer sporadic update-rate range: 30.0 – 30.0 Hz
    periodic update-rate: 40.0 Hz
    oversampling detected (curr. 40.0 > in 30.0 Hz)
}

ServiceRef FrontLaser.LaserScanServer periodic update-rate: 10.0 Hz
undersampling detected (curr. 10.0 < in 40.0 Hz)

CauseEffectChain PlannedNavigationLoop {
    ServiceRef Base.BaseStateServer sporadic update-rate range: 30.0 – 30.0 Hz
    periodic update-rate: 40.0 Hz
    oversampling detected (curr. 40.0 > in 30.0 Hz)
}

ServiceRef Mapper.MapServer periodic update-rate: 4.0 Hz

ServiceRef Planner.GoalServer periodic update-rate: 4.0 Hz

ServiceRef ObstacleAvoidance.VelocityServer periodic update-rate: 10.0 Hz
oversampling detected (curr. 10.0 > in 4.0 Hz)
```

DSLRob 2015, A. Lotz et. al.
Robotics experts focus on functional component development without presuming application specific needs.

Application experts configure the components by selecting appropriate ActivationSources within the boundaries of according ActivationConstraints and satisfying the overall E2ELatencySpecs.

E2ELatencySpecs define application-specific boundary conditions for sensor-to-actuator component chains and allow to additionally monitor and verify the actual interaction behavior of components at run-time.
Conclusion

- The choice of an ActivationSource for individual Tasks in a Component enables system integrators to select the right model of computation and thus purposefully influence the components' interaction behavior on model level.
- CauseEffectChains allow to identify relevant chains of components.
- E2ELatencySpecs allow to directly annotate application-specific constraints with respect to latencies and jitter for communication in a chain of interconnected components.
- Meta-Model core elements (such as Component, InPort, OutPort) are kept generic, thus allowing to reuse many popular component models in robotics and beyond.
- Component developers (i.e. robotics experts) and system integrators (i.e. application domain experts) can focus on their individual expertise while collaboratively designing and developing the whole robotic system (i.e. separation of roles and concerns).
Servicerobotics-lab in Ulm

www.servicerobotik-ulm.de