Generation of Flow-Preserving Orocos Implementations of Simulink/Scicos Models

Matteo Morelli and Marco Di Natale

IEEE-ICRA 2013, Workshop On Software Development and Integration in Robotics
May 6th, 2013

TeCIP Institute, Scuola Superiore Sant’Anna
Area della Ricerca CNR, Via Moruzzi, 1
56127 Pisa, ITALY
Robot Application Development Process

Key paradigms (see, e.g., BRICS FP7)

▶ Component-Based Software Engineering (CBSE)
  ▶ Components as black-boxes with interfaces
▶ Separation of Roles & Concerns
  ▶ Component Developer (Comput.)
  ▶ System Integrator (the other 4Cs)
▶ Model-Driven Engineering (MDE)
  ▶ Set of rules (meta-model) that standardize components’ interfaces and their interconnections
  ▶ Model-to-Model (M2M) & Model-to-Code (M2C) transformation engines that generate the system’s structure
Limitations – Designing Capabilities

Lack of a formal Model of Computation (MoC)

- The system-level behavior emerges from the cooperation of components (event signals)

⇒ Prevent early verification and validation (V&V) of the properties of the controller-controlled system (simulation/model-checking)

It’s difficult to realize a formal semantics (synchronous) at system-level

- Causal dependency between producers & consumers defines a partial order of execution

⇒ In general, this is not trivial to express by using event signals

Components’ functionalities are mostly handwritten; they can be generated from a model (Orocos-Simulink), but with several limitations

- Only single-task implementations can be generated
- Only for single-CPU, single-core systems
- Only implementations that preserve the synchronous assumption
What is the Synchronous Assumption?

Model simulation in Simulink

1. Model compilation & definition of one total execution order
2. Virtual time initialization
3. Block execution according to precedences & virtual time
4. Advancement of virtual clock & back to 3

Synchronous assumption

⇒ The reaction of the system (outputs & next state) must be computed before the next event in the system
Flow Preservation

In many cases, what is required from an implementation is not the preservation of the synchronous assumption but a looser property, called flow preservation

⇒ All data flows are preserved, even if results are produced at different times

▶ Possible problems because of preemption and scheduling
▶ Communication channels to be protected for data consistency
  ▶ Buffering mechanism (e.g., lock-free commn. policies)
Limitations (cont’d) – Platform Modeling

Available tools do not support execution-platform modeling, crucial for designing reliable applications

- Robot applications demand real-time tasks at different rates
- Delays and jitter may degrade the QoS and even jeopardize the control stability
  - Complex filtering (e.g. vision)
  - Bus arbitration
  - Transmission delay
  - Control action computation
- They depend on the physical execution platform: single-core, multi-core or distributed, CPU, FPGA, network protocols, ...
Proposed Approach

A system engineering process integrating MBD and MDE

- Instance of a model of concurrent tasks that exchange messages, preserving the simulation-time execution semantics and the V&V results obtained on the functional model.

- Space of the apps (Simulink/Scicos), independent from the execution hardware.

- Space of the software implementation of tasks and messages (SysML).

- Space of the execution hardware (SysML), independent from the functionality.
Focus of this talk

Proposed Approach

Functional model with back-annotations of estimated delays, to verify (by simulation) their impact on the control stability.

Model for Worst-Case Time Analysis (RTSIM, CHEDDAR, ...)

Behavioral code (Scicos)

Platform-specific implementation (PSI) (Orocos-RTT)

Simulink

Scicos

Model exporter

Simulink

Scicos

Simulink Coder/
Embedded Coder

Functional model with back-annotations of estimated delays, to verify (by simulation) their impact on the control stability.

Model for Worst-Case Time Analysis (RTSIM, CHEDDAR, ...)

Behavioral code (Simulink)

Platform-specific implementation (PSI) (Orocos-RTT)

Focus of this talk

Generation of Flow-Preserving Orocos Implementations of Simulink/Scicos Models
Functional modeling

Plant dynamics & control logics are modelled with Simulink/Scicos and tested by simulation with logical execution-time assumptions.

Main points

- Formal execution semantics, Synchronous-Reactive (SR)
- Capability of modelling and simulating FSMs
- Enable early V&V with the simulation of functionalities (& refinement of control strategies), including models of the plant
- Subsystems are units for code-generation (must be single-rate)
From Simulink/Scicos to SysML

Once the simulation results are satisfactory, the model of functional level changes from executable to a structural one (SysML).

It’s a two-step procedure

1. Import the functional model in EMF
   - An Ecore meta-model has been defined for the import process
   - A description of the Simulink/Scicos model is created by an importer that conforms to the functional meta-model (blocks, parameters, connections, ...)

![Diagram of EMF import process](image)
2. M2M transformation by QVTo

- The QVTo transform creates a Papyrus SysML model
- Subsystems, dependencies & topology of communications are generated automatically
- Standard SysML Internal Block Diagrams (IBD) can be used to represent them
Tailoring SysML: Profiles and Stereotypes

SysML is a general-purpose systems modeling language: it can (should) be customized for specific domains

- **Stereotypes** are domain-specific definitions that extend existing concepts adding properties and constraints
- **Profiles** are collections of stereotypes to be used in the context of a modeling or analysis activity
- We defined a profile collecting concepts related to functional modeling (stereotyped Blocks, FlowPorts & Dependency)
Generation of the Behavioral Code

Tools: Simulink Coder/Embedded Coder (Simulink) or E4Coder (Scicos)

Remark: (single-rate) subsystems are units for code-generation

- Each subsystem translates into a C function, the `step()`
- Each `step()` performs the output & state update of all blocks inside the subsystem (in the order imposed by the model semantics)
- Subsystem’s ports are accessed via a middleware-level API (and not as global variables)
Excution-Platform Modeling

Define the HW & SW resources available in the system

- Profiles/Stereotypes for the embedded system domain
- Definition of the physical platform (Cores, Peripherals, IODevices, Boards, ...)
- Definition of the basic software (RTOS, IODrivers, CommunicationStack, Middleware)
- Some concepts from OMG’s SysML MARTE profile, other concepts from custom definitions
A profile representing SW concurrency & resources

- Thread is the unit of concurrent execution
- Thread’s period & priority are specified by the designer
- SRSubsystems are executed in the context of Threads
- Signal variables (Comm.Imp.) represent functional communications (according to the actual mapping & deployment)
SRSubsystems must be deployed onto the execution platform

A profile for mapping (Allocated stereotype)

▶ FunctionToThreadMapping defines SRSubsystems-to-Threads mapping
  ▶ step()s executed according to the mappingOrder (subject to validation)

▶ SignalImplementationMapping maps communications to signals vars. (UDP, lock-free, etc.)

▶ ThreadToCPUDeployment is for Thread-to-CPU allocation
Mapping Modeling

It’s good practice to use multiple IBDs to define mapping models
Orocos-RTT as Execution-Platform

From the mapping model, a set of M2M & M2C transforms generate the flow-preserving, platform-dependent code for threads & commns.

A profile for a subset of Orocos-RTT (refinement)

- Static deployments (stereotyped Block)
- RTT components (stereotyped Blocks)
- In/Out data-flow, In event ports (stereotyped FlowPorts)
- Connection policy (stereotyped Connector)
1. M2M transform by QVTo

- The abstract concept of Thread is represented by the pair \textit{Activity}/\textit{TaskContext}
- Activities are non-periodic
- Inter-component communications are represented by data-flow ports on the components & lock-free \textit{ConnectionPolicy} objects
- Model elements are generated to implement scheduling strategies preserving the functional model semantics

\Rightarrow A model of the PSI
2. M2C transform by Acceleo

- **Task-code** (updateHook() as a sequence of calls to the step()s mapped on it & serialized according to the mapping order)
- **Code implementation of the signal variables** (local, inter-component communications, etc)
- **Code implementation of the scheduling & event infrastructure** that guarantees the execution order among components

⇒ The Code of the PSI
Scheduling Strategies for Flow Preservation

- Single-core platforms
  - One additional TaskContext generated as a dispatcher
  - It runs **periodically** at the base period at the **highest** priority
  - A **scheduling table** specifies the order of exec. of non-periodic components in the hyperperiod
  - They are added as **peers** to the dispatcher that **triggers** them according to the sched. table

- Single-CPU, multi-core platforms
  - One dispatcher for core
  - **Enforcement** of precedence constraints by **event signals**
Conclusions

The current approach to SW develop. in robotics has limitations:

- Lack of a formal MoC that enables MBD and V&V of controls
- No support for the definition of concepts & models of physical execution platforms (HW/basic SW)

We described a design flow where MBD (Simulink/Scicos) & MDE (SysML) are used in a complementary way, enabling:

- Separation of concerns (functional & platform models)
- Automatic code-generation of flow-preserving PSI (Orocos-RTT) of functional algorithms, verified earlier by, e.g., simulation
- Evaluation of computation and communication delays & their impact, e.g., on the task-set schedulability & the control stability and that leverages standards (MOF, SysML, MARTE, M2M, M2C)

Future work concerns

- Extension to models & examples for distributed architectures
- Estimation of delays & automated synthesis of optimal implementations (according to non-functional metrics)
thank you!

matteo.morelli@sssup.it