Real-Time Systems

Part 8: Model-Driven Design
Content

• Model building
• Model-driven design of real-time systems
• Dynamic models
• Models of computation
• Tools
  – UML
  – Ptolemy
  – Esterel
  – MATLAB/Simulink
Literature

- Sastry et al: Scanning the issue – special issue on modeling and design of embedded software, Proceedings of the IEEE, vol.91, no.1, pp. 3-10, Jan 2003
- Ptolemy: Software and Documentation
  [http://ptolemy.eecs.berkeley.edu/ptolemyII/index.htm](http://ptolemy.eecs.berkeley.edu/ptolemyII/index.htm)
- Publications on Esterel, Lustre, Safe State Machines:

Publications of IEEE, Springer, ACM can be downloaded inside the university network through the TUM Informatics Proxy (proxy.in.tum.de)
Model

- **Britannica:**
  Scientific modeling, the generation of a physical, conceptual, or mathematical representation of a real phenomenon that is difficult to observe directly. Scientific models are used to explain and predict the behaviour of real objects or systems and are used in a variety of scientific disciplines, […]

- **Wikipedia:**
  In the most general sense, a model is anything used in any way to represent anything else. [...] a conceptual model is a model that exists only in the mind. Conceptual models are used to help us know and understand the subject matter they represent.
Modeling, Design and Analysis

- **Modeling:**
  - Model is a *representation of reality*
    (usually limited to the essentials)
  - Mirroring system properties
  - What the system is doing

- **Design:**
  - Implementation
  - Hardware/Software
  - How the system is producing the required results

- **Analysis:**
  - Understanding the system
  - Why the system is doing what it is supposed to
Modeling

- Continuous Dynamics
- Discrete Dynamics
- Hybrid Systems
- State Machines
- Concurrent Models of Computation
Modeling

- As simple as possible, as complex as necessary!
  (Occam’s Razor)

- Models can be built for:
  - The *environment* for a better understanding of the processes to be controlled
  - The underlying (electronic) *hardware* to understand exact timing issues
  - The *software* to be developed to refine specification & generate code
Model-driven design of Real-time systems

- Development of Real-time systems calls for:
  - **Guaranteed** performance
  - **Fault-tolerant** systems
  - **Verification** against a model

➡ Models of hardware, software and the environment are necessary!

- Only a model makes formal verification possible!
- A **functional model** is an executable description.
Model-driven design of Real-time systems

• Models can take different roles along the development process:
  – Design
  – Simulation
  – Code generation
  – Analysis

• Some tools can cover the whole process, turning the development process into a graphical programming approach
Model-driven design – Advantages

• Advantages:
  – Models are often easier to understand than programming code (graphical representation, higher level of abstraction)
  – Systems can be simulated, evaluating different designs
  – Code generation can reduce the time to implement changes
    This is only true if the process is fully automatic (inconsistencies!)
  – Formal methods can be applied to test the code against the models

• Disadvantages:
  – Underlying processes are less understood and can still lead to unwanted behavior
Continuous Dynamics

- Modeling physical systems
  - Mechanics (motion dynamics)
  - Electrical systems
  - Thermodynamics
- System of first-order ordinary differential equations (ODE)
  \[ \dot{x} = f(x, u) \]
- Usage:
  - Simulation of environment
  - Controller development

*Springer Handbook of Robotics, 2008*
Continuous Dynamics – Example

- Motor Dynamics:

\[ u = R_A i_A + L_A \frac{di_A}{dt} + u_{emf} \]
\[ u_{emf} = c_M \omega_M \]
\[ \tau_A = c_M i_A \]
\[ \tau_A = J_M \dot{\omega}_M + \tau \]

- Direct integration of ODEs is used to simulate motor dynamics
- Any system of ODEs can be expressed in e.g. MATLAB/Simulink for simulation and controller development
Dynamic Models – Discrete Dynamics

• Discrete dynamical systems are discrete in time

• Any continuous system, expressed as a system of ODEs, can be discretized with a timestep T

• The new state vector $x_k$ is expressed as a function of the previous states $x_{k-1} - x_{k-n}$ and the current input $u_k$

\[
x_k = f(x_{k-1}, x_{k-2}, \ldots, x_{k-n}, u_k)
\]

• Example – Discrete Integrator:

\[
x_k = x_{k-1} + Tu_k
\]

• Usage:
  – Digital control design
Finite State Machines (FSM)

- Discrete in states (time is irrelevant)
- Defined as a (finite) set of states $S = \{s_1, s_2, \ldots, s_n\}$ and transitions which are activated by a guard expression
- Each transition leads to an action
- Description of a logical sequence

- Purpose in software engineering:
  - Code generation
  - Verification

Lee, Seshia, 2011
Hybrid Systems

- Composite of FSMs with continuous systems
- The set of ODEs used to express the system is selected depending on the state of the FSM
- Usage: model physical systems

\[ \begin{align*}
    y_1(t) &= y_2(t) \\
    y(t) &= y_1(t) \\
    \dot{y}(t) &= (\dot{y}_1(t)m_1 + \dot{y}_2(t)m_2)/(m_1 + m_2) \\
    \dot{y}_1(t) &= k_1(p_1 - y_1(t))/m_1 \\
    \dot{y}_2(t) &= k_2(p_2 - y_2(t))/m_2 \\
    \dot{y}(t) &= \frac{k_1p_1 + k_2p_2 - (k_1 + k_2)y(t)}{m_1 + m_2} \\
    y_1(t) &= y(t) \\
    y_2(t) &= y(t)
\end{align*} \]
Models of Computation – Reactive Systems

• Reactive systems (also: reflex systems) produce an output $o$ for each input event $i$

• Reactive systems are used e.g. in industrial process control or for control of airplanes and cars.

• Emphasis is placed on safety and determinism

• Execution of events can overlap

![Diagram of reactive systems reaction to events](image-url)
Models of Computation – Synchronous Reactive (SR) Systems

• The *synchrony hypothesis* assumes that the underlying physical machine is infinitely fast.

  ➢ Ideal assumption of a system producing outputs synchronously to changes of the inputs.

  ➢ Reaction intervals are reduced to reaction instants

• *Justification*: This assumption is correct when the probability of a second event happening during the execution of a first event drops towards zero.

• Examples:
  – Esterel
  – SCADE/Lustre
  – SystemC
Concurrent Models of Computation

• Dataflow
  – Execution is driven by data
  – If one actor needs data produced by another actor, execution has to wait until data is ready
  – Usage: Execution of discrete dynamics

• Time-Triggered Execution
  – Execution is planned for each instant in time
  – Usage: Real-time control
  – Tools: Giotto, FTOS

• Component Interaction:
  – Composition of data and query driven execution
  – Example: Web Server
Concurrent Models of Computation

• Process Networks
  – Processes communicate through channels
  – Channels store messages in a queue (asynchronous messages)
  – Usage: distributed systems

• Rendezvous
  – Processes communicate via synchronous messages (Processes wait until both sender and receiver is ready)
  – Examples: CSP, CCS, Ada
Tools – UML

• Unified Modeling Language (UML)
• Inherent part of the software development process

• Problems:
  – Heterogeneous models
  – No model of computation, therefore no code generation (except FSM)
  – Inconsistencies between model and code
Tools – UML

• Useful executable modeling languages impose *constraints* on the designer.

• The constraints may come with benefits:
  – Model is constrained to the fundamental part
  – Models may be (partly) verifyable
  – Code generation

We have to stop thinking of constraints as a universal negative!!!
Tools – Ptolemy

- Ptolemy-Project at UC Berkeley
- Named after **Claudius Ptolemaeus**
- Support of several different models of computation
- Ptolemy supports:
  - Modeling
  - Simulation
  - Code generation
  - Formal verification (to some extent)

- Information and download: [http://ptolemy.eecs.berkeley.edu/](http://ptolemy.eecs.berkeley.edu/)
Tools – Ptolemy: Actor Oriented Design

- Ptolemy models use actors instead of objects
- Objects:
  - Focus on control flow
  - System manipulates objects
- Actors:
  - Focus on data flow
  - Actors manipulate the system
- Both paradigms increase reusability by dividing the system into subsystems
- Actors can be used to represent parallelism of actions
Tools – Ptolemy: Actor Oriented Design

• Ptolemy implements several models of computations

• User can define the model of computation used by choosing a so called director.
  
  ➢ Separation of the logical and temporal design by separating the director choice from the actor connections

• Which model of computation to choose, depends on the use case

• Subsystems can be represented with different directors
  
  ➢ Representation of different domains in a single model:
    – Environment
    – Hardware
    – Software
Example Ptolemy Model of Computation: Synchronous Dataflow

- **General principle:**
  - Assumption: infinitely fast machine
  - Data is processed periodically
  - Data flow is executed once per period

- **Advantages:**
  - Static memory allocation
  - Static schedule
  - Deadlocks are detectable
  - Runtime can be calculated

- **Tools:**
  - Matlab
  - Labview
  - EasyLab
Example Ptolemy Model of Computation: Synchronous Reactive

- General principle:
  - Assumption: infinitely fast machine
  - Discrete Events (DE) are used periodically (Events do not have to occur each period)
  - One reaction per round
  - Often used with Finite State Machines

- Advantages:
  - Easy formal verification

- Tools:
  - Esterel Studio
  - SCADE/Lustre
Example Ptolemy Model of Computation: Discrete Event

- **General principle:**
  - Communication through events
  - Each event has a time stamp and a value

- **Usage:**
  - Digital Hardware
  - Telecommunication

- **Tools:**
  - VHDL
  - Verilog
Example Ptolemy Model of Computation: Continuous Time

- **General principle:**
  - Continuous signals (actors represent ODEs)

- **Usage:**
  - Simulation

- **Tools:**
  - MATLAB/Simulink
  - Labview

![Diagram of a Ptolemy model of computation](image.png)
Tools – Esterel

• Esterel is actually more a programming language than a modeling language, however it uses the *Synchronous Reactive* model of computation

• Esterel was developed by Jean-Paul Marmorat and Jean-Paul Rigault to meet the challenges of real-time systems:
  – Expression of parallelism and preemption
  – Strict concept of timing

• G. Berry developed the formal semantics of Esterel

• There are code generators to generate e.g. sequential C, C++ Code:
  – In Esterel (parallel) programs are transformed into a *single* finite state machine
  – The FSM is converted into a program with a *single* process
    ➢ Deterministic execution can be proven despite a parallel model

• Example: SCADE (a commercial tool that uses Esterel) was used to develop components for the Airbus A380.

• An Esterel compiler is freely available at [http://www-sop.inria.fr/esterel.org/files/](http://www-sop.inria.fr/esterel.org/files/)
Introduction to Esterel

- Esterel is a *synchronous language*. These languages were developed to program reactive systems.
  
  Other Examples:
  - Lustre
  - Signal
  - Statecharts

- **Reactive Systems** directly generate *output reactions* for *input events*
  - *Interactions* with the environment are the main building blocks of the system
  - Physical time is replaced by a *notion of order* (of the occurring events)
  - Interactions with the environment (*macro steps*) consist of sub steps (micro steps).
Tools – Esterel – Determinism

• Esterel is deterministic: a sequence of input events (even simultaneous ones) will always produce the same sequence of output events

• Each Esterel instruction and construct is guaranteed to be deterministic. This is achieved by
  – the constraints the language puts on the designer
  – the compiler that is proven to produce deterministic code

• Determinism of the language majorly simplifies the verification of applications
Tools – Esterel – Basics

• Communication is achieved through signals and sensors
  – Sensors provide measurements; They always provide a value (independent of changes)
  – Signals fire whenever there is an event; This can be used for I/O operations

• There are two kinds of signals
  – Non-valued signals (signal is either present or not)
  – Valued signals (signal contains a value that can be used by the consumer)

• Esterel programs can be divided into modules

• Communication between modules is achieved through a broadcast mechanism
Tools – Esterel – Basics

- Signal statements
  - `emit`: Sends a signal
  - `await`: Waits until the specified signal is present
  - `present`: Tests if a signal is present

- Execution of a module can be aborted by calling `abort`:
  - Syntax: `abort Body when Exit.Condition`

- Periodic execution can be achieved by the `every` statement:
  - Syntax: `every Occurrence do Body end every`

- Composition of instructions:
  - Blocks of commands (Modules) can be executed `sequentially` or in `parallel`.
  - Blocks of commands (Modules) can be executed `repeatedly`.
  - Blocks of commands (Modules) can be `interrupted`.
Tools – Esterel – Example: Temperature Control
Tools – Esterel – Example: Temperature Control

- Goal: Temperature control (operating range: 5 – 40°C) with a simple controller

- Operating mode:
  - Whenever the temperature comes to be close to one of the limits, the heater or ventilation is turned on, respectively.
  - If the temperature stays high (or low), the ventilation (or heating) is set to strong mode
  - When control reaches the normal temperature range, the ventilation or heating is turned off
  - If the temperature leaves the operating range, the module sends an abortion signal
Tools – Esterel – Example: Temperature Control

module TemperatureController:
input TEMP: integer, SAMPLE_TIME, DELTA_T;
output HEATER_ON, HEATER_ON_STRONG,
HEATER_OFF, VENTILATOR_ON, VENTILATOR_OFF,
VENTILATOR_ON_STRONG, SIG_ABORT;

relation SAMPLE_TIME => TEMP;

signal COLD, NORMAL, HOT in
every SAMPLE_TIME do
  await immediate TEMP;
  if ?TEMP<5 or ?TEMP>40 then emit SIG_ABORT
  elseif ?TEMP>=35 then emit HOT
  elseif ?TEMP<=10 then emit COLD
  else emit NORMAL
  end if
end every

loop
  await
  case COLD do
  emit HEATER_ON;
  abort
  await NORMAL;
  emit HEATER_OFF;
  when DELTA_T do
  emit HEATER_ON_STRONG;
  await NORMAL;
  emit HEATER_OFF;
  end abort
  case HOT do
  %...
  end await
end loop
end signal
end module
Tools – MATLAB/Simulink

- Synchronous block diagram environment
- Models of computation:
  - Continuous dynamics
  - Discrete dynamics
  - Extensions through toolboxes available
- Features:
  - System-level design
  - Simulation
  - Extendable by MATLAB and C/C++ algorithms
  - Extendable multitarget Code generation
Tools – MATLAB/Simulink

• Usage:
  – Simulation of physical processes
  – Controller design
  – Prototyping (e.g. dSpace)
  – Code generation (Real-time workshop toolbox)
  – Verification and Validation (toolbox)
  – HIL/SIL
Tools – MATLAB/Simulink – Example

- Simulation of continuous dynamic systems (Example of the DC motor model),
- Models can be built and extended quickly
- Simulation with different ODE solvers
Tools – MATLAB/Simulink – Example

- Controller design & test
- Code generation of controller part for different platforms
Tools – MATLAB/Simulink – HIL/SIL

- Hardware in the Loop (HIL)
- Software in the Loop (SIL)

- Embedding the embedded system (hardware or software) into a simulation for testing purposes