Finite state machines & message passing: SDL

Peter Marwedel
TU Dortmund,
Informatik 12

2011/11/02

These slides use Microsoft clip arts. Microsoft copyright restrictions apply.
Context

<table>
<thead>
<tr>
<th>Communication/Computation</th>
<th>Communication/Computation</th>
<th>Message passing</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Shared memory</td>
<td>blocking</td>
</tr>
<tr>
<td>FSM</td>
<td>StateCharts</td>
<td>Non-blocking</td>
</tr>
</tbody>
</table>

SDL used here as a (prominent) example of a model of computation based on asynchronous message passing communication.

appropriate also for distributed systems
SDL

Language designed for specification of distributed systems.

- Dates back to early 70s,
- Formal semantics defined in the late 80s,
- Defined by ITU (International Telecommunication Union): Z.100 recommendation in 1980
SDL

- Provides textual and graphical formats to please all users,
- Just like StateCharts, it is based on the CFSM model of computation; each FSM is called a process,
- However, it uses message passing instead of shared memory for communications,
- SDL supports operations on data.
SDL-representation of FSMs/processes

Process P1

state
input
output
Communication among SDL-FSMs

Communication between FSMs (or “processes“) is based on **message-passing**, assuming a **potentially indefinitely large FIFO-queue**.

- Each process fetches next entry from FIFO,
- checks if input enables transition,
- if yes: transition takes place,
- if no: input is ignored (exception: SAVE-mechanism).
Determinate?

Let tokens be arriving at FIFO at the same time:

Order in which they are stored, is unknown:

All orders are legal: simulators can show different behaviors for the same input, all of which are correct.
Operations on data

Variables can be declared locally for processes. Their type can be predefined or defined in SDL itself. SDL supports abstract data types (ADTs). Examples:

```
DCL
Counter Integer;
Date String;
```

```
Counter := Counter + 3;
```

```
(1:10) (11:30) ELSE
```
Process interaction diagrams

Interaction between processes can be described in process interaction diagrams (special case of block diagrams). In addition to processes, these diagrams contain channels and declarations of local signals. Example:

```
BLOCK B1

Signal A, B;

process P1

[A, B] [A, B]
Sw1
Sw2

process P2

[A]
```

Hierarchy in SDL

Process interaction diagrams can be included in blocks. The root block is called system.

Processes cannot contain other processes, unlike in StateCharts.
Timers can be declared locally. Elapsed timers put signal into queue (not necessarily processed immediately). RESET removes timer (also from FIFO-queue).
Additional language elements

SDL includes a number of additional language elements, like

- procedures
- creation and termination of processes
- advanced description of data
- More features added for SDL-2000 (not well accepted)
Application: description of network protocols

System

Processor A  Router  Processor B  Processor C

C1  C2  C3

Block Processor A

layer-n

.....

layer-1

Block Processor B

layer-n

layer-1

.....

layer-1

Block Processor C

layer-n

layer-1
Larger example: vending machine

Machine° selling pretzels, (potato) chips, cookies, and doughnuts:
accepts nickels, dime, quarters, and half-dollar coins.
Not a distributed application.

Overall view of vending machine

**System VendingMachine**
- Ccoins
  - [nickel, dime, quarter, half]
  - CoinInterface
- Crequest
  - [pur_pretzel, pur_chip, pur_cookie, pur_doughnut, reload_pretzel, reload_chip, reload_cookie, reload_doughnut]

**ChangeInterface**
- SIGNAL
  - [dime, nickel, quarter, half, pur_pretzel, pur_cookie, pur_doughnut, pur_chip, add(int), spit_change(int), amount_entered(int), reject_further_coins, exact_only, accept_coins, reject_coins, spit_dime, spit_nickel, pretzel_empty, spit_pretzel, chip_empty, spit_chip, cookie_empty, spit_cookie, doughnut_empty, spit_doughnut, reload_pretzel, reload_chip, reload_cookie, reload_doughnut]

**DecodeRequests**
- Cadd
  - [add]
  - Ccoinctrl
    - [rej_further_coins, accept_coins]

**Reject**
- [reject_coin]
  - CamontDisplay
    - [amount_entered]
  - Cemptydisplay
    - [pretzel_empty, chip_empty, cookie_empty, doughnut_empty]
  - CspitPurchased
    - [spit_pretzel, spit_chip, spit_cookie, spit_doughnut]

**Change**
- [spit_change]

**CspitChange**
- [spit_nickel, spit_dime]

**CexaktDisplay**
- [exact_only]
Decode Requests

CONNECT Cadd AND Radd;
CONNECT Cointrol AND Rcointrol;
CONNECT Cchange AND Rchange;
CONNECT CAmountDisplay AND RamountDisplay;
CONNECT Crequest AND Rpretzel,Rchip,Rcookie,
    Rdoughnut;
CONNECT CemptyDisplay AND Rpretzel_e,Rchip_e,
    Rcookie_e,Rdoughnut_e;
CONNECT CspitPurchased AND Rpretzel_s,
    Rchip_s,Rcookie_s,Rdoughnut_s;

SYNONYM PRETZEL int=50
SYNONYM PCHIP int=15;
SYNONYM PCOOKIE int=55;
SYNONYM PDOUGHNUT int=60;
SYNONYM PMAX int=60;
SYNONYM NITEMS items=7;

SIGNAL sub(int);
ChipHandler

Process ChipHandler

DCL nchip items:=NITEMS;
VIEWED current int;

pur_wait

pur_chip

VIEW(current) >= PCHIP

no

pur_wait

yes

sub(PCHIP)

nchip:= nchip-1;

spit_chip

nchip=0

yes

chip_empty

empty

reload_chip

nchip:=NITEMS

pur_wait

no

pur_wait
Evaluation & summary

- FSM model for the components,
- Non-blocking message passing for communication,
- Implementation requires bound for the maximum length of FIFOs; may be very difficult to compute,
- Excellent for distributed applications (used for ISDN),
- Commercial tools available (see http://www.sdl-forum.org)
- Not necessarily determinate
- Timer concept adequate just for soft deadlines,
- Limited way of using hierarchies,
- Limited programming language support,
- No description of non-functional properties,
- Examples: small network + vending machine
- Becoming less popular
Data flow models

Peter Marwedel
TU Dortmund,
Informatik 12

2011/11/02


These slides use Microsoft clip arts. Microsoft copyright restrictions apply.
Data flow as a “natural” model of applications

Example: Video on demand system

www.ece.ubc.ca/~irenek/techpaps/vod/vod.html
Data flow modeling

Definition: Data flow modeling is … “the process of identifying, modeling and documenting how data moves around an information system. Data flow modeling examines

- processes (activities that transform data from one form to another),
- data stores (the holding areas for data),
- external entities (what sends data into a system or receives data from a system, and
- data flows (routes by which data can flow)”.

## Models of computation considered in this course

<table>
<thead>
<tr>
<th>Communication/local computations</th>
<th>Shared memory</th>
<th>Message passing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Undefined components</td>
<td>Plain text, use cases</td>
<td>Synchronous</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Asynchronous</td>
</tr>
<tr>
<td>Communicating finite state machines</td>
<td>StateCharts</td>
<td>SDL</td>
</tr>
<tr>
<td>Data flow</td>
<td>(Not useful)</td>
<td>Kahn networks, SDF</td>
</tr>
<tr>
<td>Petri nets</td>
<td></td>
<td>C/E nets, P/T nets, …</td>
</tr>
<tr>
<td>Discrete event (DE) model</td>
<td>VHDL, Verilog, SystemC, …</td>
<td>Only experimental systems, e.g. distributed DE in Ptolemy</td>
</tr>
<tr>
<td>Von Neumann model</td>
<td>C, C++, Java</td>
<td>C, C++, Java with libraries</td>
</tr>
<tr>
<td></td>
<td>CSP, ADA</td>
<td></td>
</tr>
</tbody>
</table>
Kahn process networks

- Each component is a program/task/process, not an FSM
- Communication is by FIFOs; no overflow considered
  - writes never have to wait,
  - reads wait if FIFO is empty.
- Only one sender and one receiver per FIFO
  - no SDL-like conflicts at FIFOs
Example

Process f(in int u, in int v, out int w) {
    int i; bool b = true;
    for (;;) {
        i = b ? wait(u) : wait(v);
        //wait returns next token in FIFO, waits if empty
        send(i, w);  //writes a token into a FIFO w/o blocking
        b = !b;
    }
}
Properties of Kahn process networks

- Communication is only via channels;
- Mapping from $\geq 1$ input channel to $\geq 1$ output channel;
- Channels transmit information within an unpredictable but finite amount of time;
- In general, execution times are unknown.
Key beauty of KPNs

- A process cannot check whether data is available before attempting a read.
- A process cannot wait for data for more than one port at a time.
- Therefore, the order of reads depends only on data, not on the arrival time.
- Therefore, Kahn process networks are determinate (!); for a given input, the result will always the same, regardless of the speed of the nodes.

Many applications in embedded system design: Any combination of fast and slow simulation & hardware prototypes always gives the same result.
Computational power and analyzability

- KPNs are Turing-complete (anything which can be computed can be computed by a KPN)
- It is a challenge to schedule KPNs without accumulating tokens
- KPNs are computationally powerful, but difficult to analyze (e.g. what’s the maximum buffer size?)
- Number of processes is static (cannot change)
More information about KPNs

- See also S. Edwards: http://www.cs.columbia.edu/~sedwards/classes/2001/w4995-02/presentations/dataflow.ppt
Less computationally powerful, but easier to analyze:

**Synchronous data flow (SDF).**

- **Synchronous**
  - = global clock controlling “firing” of nodes

- Again using asynchronous message passing
  - = tasks do not have to wait until output is accepted.
(Homogeneous-) Synchronous data flow (SDF)

- Nodes are called **actors**.
- Actors are **ready**, if the necessary number of input tokens exists and if enough buffer space at the output exists.
- Ready actors can **fire** (be executed).
- Execution takes a **fixed, known time**.

Actually, this is a case of **homogeneous** synchronous data flow models (HSDF): # of tokens per firing the same.
(Non-homogeneous-) Synchronous data flow (SDF) (1)

In the general case, a number of tokens can be produced/consumed per firing; firing rate depends on # of tokens …

A ready, can fire
(Non-homogeneous-) Synchronous data flow (SDF) (2)

In the general case, a number of tokens can be produced/ consumed per firing; firing rate depends on # of tokens …

B ready, can fire
(Non-homogeneous-) Synchronous data flow (SDF) (3)

In the general case, a number of tokens can be produced/ consumed per firing; firing rate depends on # of tokens …

A ready, can fire
(Non-homogeneous-) Synchronous data flow (SDF) (4)

In the general case, a number of tokens can be produced/ consumed per firing; firing rate depends on # of tokens …

B ready, can fire
(Non-homogeneous-) Synchronous data flow (SDF) (5)

In the general case, a number of tokens can be produced/ consumed per firing; firing rate depends on # of tokens …

B ready, can fire
(Non-homogeneous-) Synchronous data flow (SDF) (6)

In the general case, a number of tokens can be produced/consumed per firing; firing rate depends on # of tokens ...

1 period complete, A ready, can fire
Actual modeling of buffer capacity

The capacity of buffers can be modeled easier:

The buffer capacity of an edge can be modeled as a backward edge where (initial number of tokens = buffer capacity).
Multi-rate models & balance equations (one for each channel)

```
fire A {
  ...
  produce N
  ...
}
```

channel

```
fire B {
  ...
  consume M
  ...
}
```

channel

\[ f_A N = f_B M \]

number of tokens consumed

number of tokens produced

number of firings per “iteration”

Schedulable statically
In the general case, buffers may be needed at edges.

Decidable:
- buffer memory requirements
- deadlock

Adopted from: ptolemy.eecs.berkeley.edu/presentations/03/streamingEAL.ppt
Parallel Scheduling of SDF Models

SDF is suitable for automated mapping onto parallel processors and synthesis of parallel circuits.

Many scheduling optimization problems can be formulated. Some can be solved, too!

Source: ptolemy.eecs.berkeley.edu/presentations/03/streamingEAL.ppt
Expressiveness of data flow MoCs

CSDF = Cyclo static data flow (rates vary in a cyclic way)

[S. Stuijk, 2007]
The expressiveness/analyzability conflict

- Kahn process networks
- SDF
- Homogeneous SDF (HSDF)

Expressiveness and succinctness

Analyzability

Implementation efficiency

[S. Stuijk, 2007]
Semantics? “Simulink uses an idealized timing model for block execution and communication. Both happen infinitely fast at exact points in simulated time. Thereafter, simulated time is advanced by exact time steps. All values on edges are constant in between time steps.” [Nicolae Marian, Yue Ma]
Threads are Not the Only Possibility: 6th example: Continuous-Time Languages

Typical usage pattern:
- model the continuous dynamics of the physical plant
- model the discrete-time controller
- code generate the discrete-time controller

Simulink + Real-Time Workshop
Starting point for “model-based design”

/* Switch: `<Root>/SW2` incorporates:
 * Sum: `<Root>/Sum1`
 * Gain: `<Root>/G1`
 * Sum: `<Root>/Sum2`
 * Gain: `<Root>/G3`
 */

for(i1=0; i1<10; i1++) {
    if(rtU.In1[i1] * 3.0 >= 0.0) {
        rtb_SW2_c[i1] = rtU.In1[i1] - rtDWork.Delay_DSTATE[i1];
    } else {
        rtb_SW2_c[i1] = (rtDWork.Delay_DSTATE[i1] - rtU.In1[i1]) * 5.0;
    }

    /* Outport: `<Root>/Out1` */
    rtY.Out1[i1] = rtb_SW2_c[i1];

    /* Update for UnitDelay: `<Root>/Delay` */
    rtDWork.Delay_DSTATE[i1] = rtb_SW2_c[i1];
}
Threads are Not the Only Possibility: 5th example: Instrumentation Languages

e.g. LabVIEW, Structured dataflow model of computation
Actor languages

The established: Object-oriented:
- class name
- data
- methods

What flows through an object is sequential control
Things happen to objects

The alternative: Actor oriented:
- actor name
- data (state)
- parameters
- ports

Actors make things happen
What flows through an object is streams of data

Input data  Output data
Summary

Data flow model of computation

- Motivation
- Kahn process networks
- SDF
- Visual programming languages
  - Simulink