Workflow Charts and Their Semantics Using Abstract State Machines

Theodorich Kopetzky
+43 7236 3343 870
theodorich.kopetzky@scch.at
www.scch.at

Verena Geist
+43 7236 3343 817
Verena.geist@scch.at
www.scch.at
Overview

- Problem Statement & Motivation
- Workflow Charts
- Abstract State Machines
- Semantics
- Implementation
- Conclusion
Problem Statement

- Different foci of languages:
  - Control flow of business processes
  - Subjects and tasks
- Modelling of user interaction often not emphasised

- Workflow charts explicitly describe user interaction by submit/response-style dialogues as typed, tripartite state machines
  - Basis: Formcharts (support for single-user scenarios of two-staged HCI without concurrency)
  - Limitation: Lack of a formal semantics
Specifying the semantics of workflow charts using the formalism of Abstract State Machines (ASMs) provides an operational semantics.

Reasons for formalisation:
- Reasoning and Simulation
- Avoiding problems arisen from the lack of a formal specification of modelling languages (cf. BPMN)
Workflow Charts
Underlying Structure

Starting DSA
Flow conditions
- unique choice
  h
Actions for modifying
system state
Type void
- Role s

CP A
Actions for reporting
- Enabling conditions
  s1
... en
Type TA

ISA B1
Activation conditions
- multiple choice 0..n
s1
... sn
Actions for modifying
system state
Type TB1

ISA

ISA Bn
Activation conditions
- multiple choice
s1
... sn
Actions for modifying
system state
Type TBN

DSA C1
Flow conditions
- unique choice
f1
... fn
Actions for modifying
system state
Type D1
- Role r1

DSA

CP D1
Actions for reporting
- Enabling conditions
  ...- en
Type D1

CP Dp
Actions for reporting
- Enabling conditions
  ...- en
Type Dp

CP Cm
Actions for reporting
- Enabling conditions
  ...- en
Type E1

CP Ek
Actions for reporting
- Enabling conditions
  ...- en
Type Ek

© Software Competence Center Hagenberg GmbH
Abstract State Machines
Game of Life

- An infinite two-dimensional orthogonal grid of square cells, each alive or dead.
- Every cell interacts with its eight neighbours (horizontally, vertically, or diagonally adjacent).
- At each step in time, the following transitions occur:
  - Any live cell with fewer than two live neighbours dies, as if caused by under-population.
  - Any live cell with two or three live neighbours lives on to the next generation.
  - Any live cell with more than three live neighbours dies, as if by overcrowding.
  - Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.
letDie(cell, n) =
    if status(cell)=Alive and (n<2 or n>3) then
        status(cell) := Dead

letLive(cell, n) =
    if status(cell)=Dead and n=3 then
        status(cell) := Alive

Rule gameOfLife =
    forall cell in allCells do
        letDie(cell, numberOfAliveNeighbors(cell))
        letLive(cell, numberOfAliveNeighbors(cell))
Abstract State Machines

- Finite set of association rules of the form
  \[
  \text{if } \text{Condition} \text{ then } \text{Updates}
  \]
  - Condition: predicate logic formula, evaluating to \text{true} or \text{false}
  - Updates: finite set of assignments of the form \( f(a_1, \ldots, a_n) := v \)

- ASM \textit{state} – classical notion of mathematical structures
  (data as abstract objects)

- ASM \textit{run} – classical notion of computation of transition systems
  (simultaneous execution of updates of rules whose guard is \text{true} regarding consistency)
Semantics of Workflow Charts

- Environment
  - Actor
    - Signalling between actors / events
  - Worklist
  - UI

- Needed definitions
  - CP_Nodes, ISA_Nodes, DSA_Nodes
  - Association relation between nodes
    - Via assoc
  - Events
  - State of an agent
Basic operation of the WFCAGENT

- waitingForSelection
- TaskFromWorklistSelected
- waitingForDataInput
- DataSubmitted
WFCAGENT =

if event = TaskFromWorklistSelected
   ^ (CtlState(self) = waitingForSelection) then
      PROCESSCLIENTPAGE(SelectedNode)
   CtlState(self) := waitingForDataInput

if event = DataSubmitted
   ^ (CtlState(self) = waitingForDataInput) then
      PROCESSEDATAINPUT(SelectedNode)
   CtlState(self) := waitingForSelection
PROCESSCLIENTPAGE(DSA) =
  seq
  action(Dsa)
  cp := EnabledNodes(Dsa; Dsa Nodes; CP Nodes)
  PRESENTUI(cp; EnabledNodes(cp; CP Nodes; ISA Nodes))
Endseq

\[
EnabledNodes(source, sourceNodeSet, targetNodeSet) = \{ target \in targetNodeSet \mid 
\begin{align*}
source & \in sourceNodeSet \land \\
\exists \ a \in Associations. source(a) = source \land target(a) = target \land guard(a)\end{align*}
\} \]
Implementation

- The Sample
- CoreASM
- Excerpts from the Source
- Test Run
Two agents

Provides tasklist selection, User input simulation

User Emulator → WfcAgent
universe UserActor = \{ U1, U2, U3 \}
universe Node = \{ D1, C1, I1, I2, D2, D3, C2, I3, D4, C3, I4 \}
universe Association = \{ A1, A2, A3, A4, A5, A6, A7, A8, A9, A10, A11 \}
Set up of the ASM

\[
\begin{align*}
\text{setAssoc}(A1, D1, C1, \text{GuardTrue}) \\
\text{setAssoc}(A2, C1, I1, \text{GuardTrue}) \\
\text{setAssoc}(A3, C1, I2, \text{GuardTrue}) \\
\text{setAssoc}(A4, I1, D2, \text{GuardTrue}) \\
\text{setAssoc}(A5, I2, D3, \text{GuardTrue}) \\
\text{setAssoc}(A6, D3, C2, \text{GuardTrue}) \\
\text{setAssoc}(A7, C2, I3, \text{GuardTrue}) \\
\text{setAssoc}(A8, D2, C1, \text{GuardTrue}) \\
\text{setAssoc}(A9, I2, D4, \text{GuardTrue}) \\
\text{setAssoc}(A10, D4, C3, \text{GuardTrue}) \\
\text{setAssoc}(A11, C3, I4, \text{GuardTrue})
\end{align*}
\]

\[
\begin{align*}
\text{user4DSA}(D1) & := U1 \\
\text{user4DSA}(D3) & := U2 \\
\text{user4DSA}(D4) & := U3 \\
\ldots
\end{align*}
\]

\[
\text{rule setAssoc}(a, s, t, g) = \{
\begin{align*}
\text{source}(a) & := s \\
\text{target}(a) & := t \\
\text{guard}(a) & := g
\end{align*}
\}
\]
Main Loop of the ASM

rule WfCAgentMain = {
    onsignal rs of usSelection do {
        if (ctl_state(self) = WaitingForSelection) then {
            ProcessClientPage(data(rs))
            ctl_state(self) := WaitingForDataInput
        }
    }
    onsignal di of usDataInput do {
        if (ctl_state(self) = WaitingForDataInput) then {
            ProcessDataInput(data(di))
            ctl_state(self) := WaitingForSelection
        }
    }
}

rule ProcessClientPage(dsa) =
    seqblock
    // dsa_action()
    cp := EnabledNodes(dsa, DSA_NodeSet, CP_NodeSet)
    // cp_action()
    PresentUI(cp, EnabledNodes(head(toList(cp)), CP_NodeSet, ISA_NodeSet))
    endseqblock
rule UserEmulatorAgent = {
    if userActorState(self) = usSelection and |worklist(wfcAgent(self))|>0 then
        par
            userActorState(self) := usWait4Gui
            signal wfcAgent(self) with usSelection as s do {
                choose wli in worklist(wfcAgent(self)) do {
                    data(s) := wli
                    remove wli from worklist(wfcAgent(self))
                }
            } endpar
    if userActorState(self) = usDataInput then
        par
            userActorState(self) := usSelection
            signal wfcAgent(self) with usDataInput as di do {
                choose isalInput in userAvailableInput(self) do {
                    data(di) := isalInput
                }
            } endpar
    ... see right ...
}
onsignal gs of "GUISETUP" do {
    if userActorState(self) = usWait4Gui then {
        userActorState(self) := usDataInput
        userAvailableInput(self) := data(gs)
    }
}
CoreASM

```plaintext
// CoreASM WfC

ÃO uses */
use Standard
use Signals
use Time
use Mach
use Modularity
use SchedulingPolicies
use DebugInfo

// includes */
include "UserEmulatorAgent.casm"
// universes */
universe WfCAgent

universe UserActor = { U1, U2, U3 }
universe Node = {O1, C1, I1, I2, D2, D3, C2, I3, D4, C3, I4 }
universe Association = {A1, A2, A3, A4, A5, A6, A7, A8, A9, A10, A11 }

// enums */
enum WfCAgentState = { WaitingForSelection, WaitingForDataInput }

//functions */
function userActorState: UserActor -> UserStatus
function userAvailableInput: UserActor -> SET
```

© Software Competence Center Hagenberg GmbH
CoreASM Run

\{C1, C2, C3, I4, D1, D4, I3, D2, D3, I2, I1\}

WfCAgent for U2 = Element38831
WfCAgent for U1 = Element38839
WfCAgent for U3 = Element38847

Agent Element38839
    enabled CP : \{C1\}
    enabled ISA: \{I2, I1\}

Agent Element38839 ProcessDataInput I1
    adding to worklist D2, undef

Agent Element38839
    enabled CP : \{C1\}
    enabled ISA: \{I2, I1\}

Agent Element38839 ProcessDataInput I1
    adding to worklist D2, undef

... Agent Element38839 ProcessDataInput I2
    adding to worklist D4, U3
    adding to worklist D3, U2

Agent Element38847
    enabled CP : \{C3\}
    enabled ISA: \{I4\}

Agent Element38831
    enabled CP : \{C2\}
    enabled ISA: \{I3\}

Agent Element38831 ProcessDataInput I3
Agent Element38847 ProcessDataInput I4

[!] Run is terminated
Conclusion

- Specification as abstract state machine showed some problems in the original introduction to workflow charts
- Inclusion in an ongoing project
- Working with CoreASM was quite fine
Science is only the current state of mistakes
Eckart von Hirschhausen