Epilogue

The Introduction (Sect. 1) concluded with the statement that we view the ASM modeling and analysis method not as a stand-alone method, but as to-be-integrated into industrial-strength software development frameworks which care about high-level design and analysis. A candidate par excellence are model-based engineering (MBE) frameworks which provide tools to support some, if not all, of the activities from requirements to code. Often they provide also support for pragmatic needs, like combining textual and graphical definition techniques, tracing requirements (so that one can effectively check their complete coverage), version management, etc. So what additional value can the ASM method bring to such frameworks?

Remember the ASM method offers a) a semantically well-defined, pseudo code language that is familiar to the software practitioner, and b) a mathematically precise yet simple technique

- to **B**uild systems **accurately at any desired level of abstraction**, providing well-documented, systematically inspectable and executable system *blueprints* one can use to **E**xplain, **D**ebug (validate and verify) and maintain (**E**xplore/Extend and **R**euse) designs (**BEDER**),
- to **accurately link designs at different levels of abstraction**, supporting not only the stepwise implementation of application domain concepts (by ASM refinement), but also a rigorous experimental and mathematical analysis to debug the implementation steps (by validation and verification at any needed level), see Fig. 1.1,
- to **intellectually manage** the tremendous **complexity** of software intensive systems of today.

Coping with complexity is a well-known problem in software engineering. It triggered a huge number of industrial frameworks and proposals to turn programming into something that is closer to the *system designer's understanding* of the underlying problem (and how to solve it) than to *explaining to machines* how they should execute the solution. The above mentioned MBE

E. Börger, A. Raschke, *Modeling Companion for Software Practitioners*,
https://doi.org/10.1007/978-3-662-56641-1

approach is an example of such an endeavor and comes with extensive tool support. Among ASM's benefits are:

- **Accurate high-level descriptions of the behavior of tools**, descriptions which are rarely found in current software engineering frameworks, can be provided as ASM models. This could also improve the quality of manuals. The lack of a clear behavioral model can make it hard for the users to grasp a correct understanding of what a tool does and to use it correctly. It may hinder the interoperability of tools. Not surprisingly, the best tools come with and need extensive, time consuming (and expensive!) training programs and technical support by the tool developer.[4]
- **Inspection and debugging of designs**, not only of code,[5] can be done *at whatever level of abstraction* using ASMs, thereby contributing to make complex systems understandable and circumventing the necessarily limited expressivity of tools. It increases the chances to detect *conceptual errors* which are easily missed by code debuggers.
- **Definition of variations of tool behavior patterns** by ASM refinements can help to cope with the multitude of frameworks which offer tools with similar, often slightly different, functionality but with different interfaces. Besides the dependency on the tool vendor (vendor lock-in), this multitude may create also a language and version dependency a *general* modeling framework—in particular if open-source—better avoids. Defining ASM models for tool cores and describing their variations by refinements can change this.[6] In combination with a standardization effort for functionality and interfaces of modeling-supporting tools, this can raise the abstraction level of tools and make them better interoperable.
- **Precise conceptual support for divide-and-conquer techniques** is offered by ASM refinements, allowing one to effectively and accurately combine models which have been defined step- and component-wise, possibly passing through a hierarchy of levels of abstraction.

Although the use of ASMs and their tools has proved to be useful in various industrial applications (some of which have been mentioned in this book), existing academic tools for ASMs are only a first step. Every contribution to reach the challenging goal of an open source and where useful standardized ASM-based software development tool environment is welcome.[7]

[4] See for example the lessons learned about the use of modeling tools in various MBE projects, reported in [159, 34] with numerous further references.

[5] 'system design errors are increasingly the cause of major accidents' [182, Sect. 2.1]

[6] An example of what we refer to is the abstract object-oriented programming language semantics model in [77]. The C# [54] and Java [232] models appear as variations of that one ASM.

[7] The joint venture the first author had proposed at the end of 1999 to Siemens and Microsoft Research, to create an ASM-based software design, analysis (debugging by validation and verification), documentation and maintenance framework, could have made building software-intensive systems more reliable and more effective than it is still today.

Appendix A
Some Complete Models in a Nutshell

We reformulate here without further explanations the complete models for a few machines described in the book, the final result of their didactically motivated piecemeal construction and explanation. They are the starting point for the refinement to CoreASM executable models. We concentrate on rewriting the rules and refer for the signature, the underlying initialization and similar assumptions to the main text where the machines are introduced. We do not repeat machines whose rules have been written out as one piece in the main text, in particular the following ones:

- 1WayStopGoLightSpec (Sect. 2.1.1) with its horizontally refined model 1WayStopGoOrangeLightSpec (Sect. 2.1.3) and the data and vertically refined Timed2WayStopGoLightSpec (Sect. 2.1.4)
- PackageRouter ground model (end of Sect. 2.3.1)
- ConcurrencyPattern (end of Sect. 2.4.3),
- Atm top level machine (Fig. 2.28),
- ConcurExtremaFinding (Sect. 3.1.2),
- TerminationDetector for diffusing computations (Sect. 3.3.1.2),
- LocalSyncTransformer with SyncShell(p) and Synchronizer(p) programs (Sect. 3.3.2),
- ThreadPoolExecutor for J2SE 5.0 consisting of two submachines HandleNewTask and HandleQueuedTask (Sect. 4.2.2.2),
- ASMs for various bilateral, multilateral, asynchronous and synchronous communication patterns (Sect. 4.4),
- the S-BPM communication model with its characteristic integration of textual and graphical control-state diagram descriptions (Sect. 5.2.2).

We also do not repeat the ASM rules which specify CoreASM, with its debugger plug-in (Chap. 8), and the Control State Diagrams (Chap. 9).

E. Börger, A. Raschke, *Modeling Companion for Software Practitioners*,
https://doi.org/10.1007/978-3-662-56641-1

A.1 TerminationDetection (Sect. 3.2)

```
TerminationDetection =                                  -- main program
  StartProbe                    -- done by master if ProbeNotSuccessful
  TriggerActivation             -- spontaneously if active without token
  BecomeActive                                          -- if hasMessage
  BecomePassive                 -- spontaneously if not hasMessage
  PassToken                     -- if passive and not the master
```

A.1.1 Token Passing Components

```
StartProbe =
  if ProbeNotSuccessful and self = master then
    if self = master then tokenColor := white           -- ColorToken
      else if color(self) = black then tokenColor := black
    ForwardToken
    color(self) := white                                -- GetWhite
PassToken =
  if hasToken and mode = passive and self ≠ master then
    if color(self) = black then tokenColor := black     -- ColorToken
    ForwardToken
    color(self) := white                                -- GetWhite
```

A.1.2 Activation Related Components

```
BecomeActive =
  if Triggered then                                     -- hasMessage(self)
    mode := active
    Deactivate(Triggered)                               -- delete the trigger
BecomePassive =
  if not Triggered                                      -- not hasMessage(self)
    then SpontaneouslyDo
      if mode = active then mode := passive
TriggerActivation =
  if mode = active then SpontaneouslyDo                 -- TriggerAct
    if not hasToken then
      choose m ∈ Machine                                -- SendMessage to m
        Activate(Triggered(m))
        if number(m) > number(self) then color(self) := black
```

A.1.3 Some Auxiliary Definitions

$Machine = \{m_0, \ldots, m_{N-1}\}$ $master = m_0$ $number(m_i) = i \bmod N$
ForwardToken =
  $hasToken(\mathbf{self}) := false$ -- forward token from machine nr. $i+1$
  $hasToken(pred(\mathbf{self})) := true$ -- ... to predecessor machine nr. i
ProbeNotSuccessful iff $hasToken(master)$ **and**
  ($tokenColor = black$ **or** $color(master) = black$)
Activate($trigger$) = -- update instruction
  $trigger := true$ -- even if in parallel with Deactivate($trigger$)
Deactivate($trigger$) = -- update instruction
  $\begin{cases} trigger := false & \textbf{if not executed in parallel with Activate}(trigger) \\ \textbf{skip} & \textbf{else} \end{cases}$

A.2 VirtualProvider (Sect. 5.1)

VirtualProvider = **one of** -- main program
  ({Process, ReceiveReq, SendReq, ReceiveAnsw, SendAnsw})

A.2.1 Communication Components

SendReq = SendAnsw = -- SendPattern, see Sect. 4.4.1
  if $readyToSend(m)$ **then** -- trigger predicate $ToSend(m)$
    if $OkSend(m)$ **then**
      Send(m)
      if $AckRequested(m)$ **then**
        SetWaitCond(m)
      if $BlockingSend(m)$ **then**
        $status := awaitAck(m)$
    else
      HandleSendFailure(m, $notOkSend$)
      $readyToSend(m) := false$ -- Done(m)

ReceivePattern(m) = -- see Sect. 4.4.1
  if $Arriving(m)$ **then**
    if $ReadyToReceive(m)$ **then**
      Receive(m)
      if $ToBeAcknowledged(m)$ **then**
        Send($Ack(m), sender(m)$)
    elseif $ToBeDiscarded(m)$ **then**
      Discard(m)

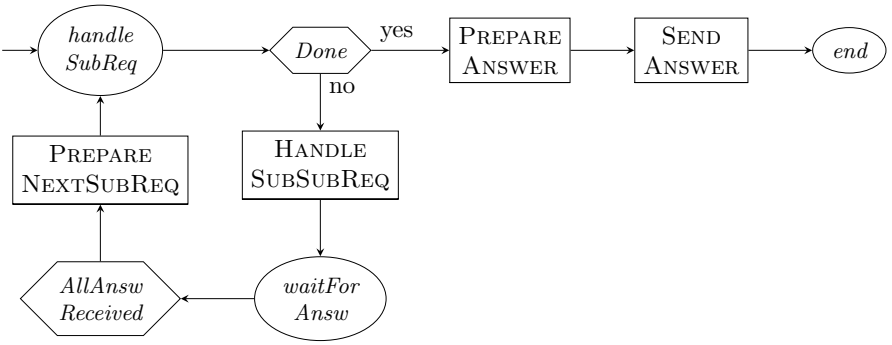
```
      if ToBeAcknowledged(m) then
        Send(Ack(m, discarded), sender(m))
    else
      if ToBeBuffered(m) then                                  -- Buffer(m)
        Insert(m, buffer)
        SetTimer(insertBuffer(m))
      if ToBeAcknowledged(m) then
        Send(Ack(m, buffered), sender(m))
  if TimeOut(insertBuffer(m)) then
    Receive(m)
    Delete(m, buffer)

ReceiveReq = ReceivePattern
  where                  -- instantiation of ReceivePattern abstractions
    if ReadyToReceive(m) then m ∈ InReqMssg
    Receive(m) =
      Consume(m)
      let r = new (ReqObj) in                          -- CreateReqObj(m)
        status(r) := start                             -- Initialize(r, m)
        reqMsg(r) := m

ReceiveAnsw = ReceivePattern
  where                  -- instantiation of ReceivePattern abstractions
    if ReadyToReceive(m) then m ∈ InAnswMssg
    Receive(m) =
      Insert(m, AnswerSet(subRequestor(m)))
      Consume(m)
```

A.2.2 Process *Component*

```
Process =
  choose r ∈ ReqObj with status(r) = start
    let a = new (Agent)                    -- CreateSubReqHandler(r)
      pgm(a) := HandleSubReq
      handler(r) := a                                  -- Initialize(a, r)
      req(a) := r
      subReq(a) := head(seqSubReq(r))                  -- current subrequest
      status(r) := handleSubReq
    AnswerSet(r) := ∅                                  -- Initialize(AnswerSet(r))
```

The HandleSubReq program with which the new subrequest handler is equipped is defined by Fig. A.1. It comes with three submachines:

Fig. A.1 HandleSubReq component of VP Process



HandleSubSubReq =
  forall $r \in (parSubReq(subReq))$ -- PrepareBroadcast
    $readyToSend(outReq2Msg(r)) := true$
  $AnswerSet(subReq) := \emptyset$ -- Initialize($AnswerSet(subReq)$)
PrepareNextSubReq =
  $subReq := next(subReq, seqSubReq(req), AnswerSet(subReq))$
  Add($AnswerSet(subReq)$, $AnswerSet(req($**self**$))$)
PrepareAnswer = **let** $req = req($**self**$)$
  $readyToSend(outAnsw2Msg(answer(req, AnswerSet(req)))) := true$

Some auxiliary definitions:

AllAnswReceived iff
  forall $q \in toBeAnswered(parSubReq(subReq))$
    thereissome $m \in AnswerSet(subReq)$ **with** $IsAnswer(m, q)$
Done **iff** $subReq = done$ -- $done \notin SubReq$

A.3 AodvSpec (Sect. 6.1)

AodvSpec = **one of** -- main program
  PrepareComm
  Router
where
  PrepareComm =
    if $WantsToCommunicateWith(destination)$ **then**
      if $KnowsActiveRouteTo(destination)$
        then
          StartCommunicationWith($destination$)

```
            WantsToCommunicateWith(destination) := false
            WaitingForRouteTo(destination) := false
          else
            if not WaitingForRouteTo(destination) then
              GenerateRouteReq(destination)
              WaitingForRouteTo(destination) := true
  Router = one of
    ProcessRouteReq
    ProcessRouteRep
    ProcessRouteErr
  ProcessRouteErr = one of
    GenerateRouteErr
    PropagateRouteErr
```

A.3.1 Route Request Components of AodvSpec

```
GenerateRouteReq(destination) =
  let r = new (RouteRequest) in
    dest(r) := destination
    destSeqNum(r) := lastKnownDestSeqNum(destination, RT)
    origin(r) := self
    originSeqNum(r) := curSeqNum + 1
    curSeqNum := curSeqNum + 1              -- Increment(curSeqNum)
    hopCount(r) := 0
    localId(r) := localReqCount + 1
    localReqCount := localReqCount + 1  -- Increment(localReqCount)
    forall n ∈ neighb do Send(r, to n)          -- Broadcast(r)
    Insert(globalId(r), ReceivedReq)            -- Buffer(r)
    if entryFor(destination, RT) ≠ undef
      then known(r) := known(entryFor(destination, RT))
      else known(r) := false

ProcessRouteReq(rreq) =
  if Received(rreq) and rreq ∈ RouteRequest then
    if not AlreadyReceivedBefore(rreq) then  -- no rreq processed twice
      Insert(globalId(rreq), ReceivedReq)
      if HasNewReverseRouteInfo(rreq) then
        BuildReverseRoute(rreq)
                -- with freshest destSeqNum for RT-entry to originator
      seq  if FoundValidPathFor(rreq)
             then  GenerateRouteReply(rreq)
             else  ForwardRefreshedReq(rreq)
    Consume(rreq)
```

BuildReverseRoute($rreq$) =
  if *ThereIsRouteInfoFor*(*origin*($rreq$), *RT*)
    -- *rreq* knows fresher *curSeqNum*(*origin*($rreq$)) than RT
    then UpdateReverseRoute(*entryFor*(*origin*($rreq$), *RT*), $rreq$)
    else ExtendReverseRoute(*RT*, $rreq$)

ExtendReverseRoute(*RT*, req) =
  let e = **new** (*RT*)
    dest(e) := *origin*(req)
    precursor(e) := ∅UpdateReverseRoute(e, req)
UpdateReverseRoute(e, req) =
  destSeqNum(e) := *originSeqNum*(req)
  nextHop(e) := *sender*(req)
  hopCount(e) := *hopCount*(req) + 1
  Active(e) := *true*
  known(e) := *true*

ForwardRefreshedReq(r) =
  let r' = **new** (*RouteRequest*)
    Copy(*dest*, *origin*, *originSeqNum*, *localId*, *known*, **from** r **to** r')
    hopCount(r') := *hopCount*(r) + 1
    destSeqNum(r') := *max*{*destSeqNum*(r), -- best known
      lastKnownDestSeqNum(*dest*(r), *RT*)} -- *destSeqNum*
    forall n ∈ *neighb* Send(r', **to** n) -- Broadcast(r')

Some auxiliary definitions:

lastKnownDestSeqNum(d, *RT*) =
  $\begin{cases} \textit{destSeqNum}(\textit{entry}) & \textbf{if forsome } \textit{entry} \in \textit{RT } \textit{dest}(\textit{entry}) = d \\ \textit{unknown} & \textbf{else} \end{cases}$

AlreadyReceivedBefore($rreq$) **iff** *globalId*($rreq$) ∈ *ReceivedReq*

HasNewReverseRouteInfo(req) **iff** req ∈ *RouteRequest* **and**
  ThereIsNoRouteInfoFor(*origin*(req), *RT*) **or**
    (*ThereIsRouteInfoFor*(*origin*(req), *RT*) **and**
      HasNewOriginInfo(req, *RT*))

HasNewOriginInfo(req, *RT*) **iff** -- info on *curSeqNum*(*origin*(req))
  let *entry* = *entryFor*(*origin*(req), *RT*)
    originSeqNum(req) > *destSeqNum*(*entry*)
    or *originSeqNum*(req) = *destSeqNum*(*entry*) **and**
      (*hopCount*(req) < *hopCount*(*entry*) **or not** *Active*(*entry*))

ThereIsRouteInfoFor(d, *RT*) **iff**
  forsome *entry* ∈ *RT* *dest*(*entry*) = d

ThereIsNoRouteInfoFor(*node*, *RT*) **iff**
  not *ThereIsRouteInfoFor*(*node*, *RT*)

FoundValidPathFor(*rreq*) **iff**
  dest(*rreq*) = **self** **or** *KnowsFreshEnoughRouteFor*(*rreq*, *RT*)

KnowsFreshEnoughRouteFor(*rreq*, *RT*) **iff forsome** *entry* ∈ *RT*
  dest(*entry*) = *dest*(*rreq*) **and** *ValidDestSeqNum*(*entry*)
    and *destSeqNum*(*entry*) ≥ *destSeqNum*(*rreq*)
      and *Active*(*entry*)

A.3.2 Route Reply Components of AodvSpec

GenerateRouteReply(*rreq*) =
  let *r* = **new** (*RouteReply*)
    dest(*r*) := *dest*(*rreq*)
    origin(*r*) := *origin*(*rreq*)
    if *dest*(*rreq*) = **self** **then**
      hopCount(*r*) := 0
      destSeqNum(*r*) := *max*{*curSeqNum*, *destSeqNum*(*rreq*)}
      curSeqNum := *max*{*curSeqNum*, *destSeqNum*(*rreq*)}
    else
      let *fwdEntry* = *entryFor*(*dest*(*rreq*), *RT*)
        hopCount(*r*) := *hopCount*(*fwdEntry*)
        destSeqNum(*r*) := *destSeqNum*(*fwdEntry*)
        PrecursorInsertion(*sender*(*rreq*), *fwdEntry*)
    Send(*r*, **to** *nextHop*(*entryFor*(*origin*(*rreq*), *RT*)))
  where
    PrecursorInsertion(*node*, *entry*) =
      Insert(*node*, *precursor*(*entry*))

ProcessRouteRep(*rrep*) =
  if *Received*(*rrep*) **and** *rrep* ∈ *RouteReply* **then**
    if *HasNewForwardRouteInfo*(*rrep*) **then**
      BuildForwardRoute(*rrep*)
      if *MustForward*(*rrep*) **then** ForwardRefreshedRep(*rrep*)
    Consume(*rrep*)

BuildForwardRoute(*rrep*) =
  if *ThereIsRouteInfoFor*(*dest*(*rrep*), *RT*)
    then UpdateForwardRoute(*entryFor*(*dest*(*rrep*), *RT*), *rrep*)
    else ExtendForwardRoute(*RT*, *rrep*)

```
ExtendForwardRoute(RT, rep) =
  let e = new (RT)          -- create new entry to extend forward route
    dest(e) := dest(rep)          -- with same destination as rep
    UpdateForwardRoute(e, rep)
UpdateForwardRoute(e, rep) =
  destSeqNum(e) := destSeqNum(rep)
  nextHop(e) := sender(rep)
  hopCount(e) := hopCount(rep) + 1
  Active(e) := true
  known(e) := true
  SetPrecursor(rep, e)

SetPrecursor(rep, e) =
  if MustForward(rep) then
    Insert(nextHop(entryFor(origin(rep), RT)), precursor(e))

ForwardRefreshedRep(rep) =
  let rep' = new (RouteReply)
    Copy(dest, destSeqNum, origin, from rep to rep')
    hopCount(rep') := hopCount(rep) + 1
    Send(rep', to nextHop(entryFor(origin(rep), RT)))
```

Some auxiliary definitions:

```
HasNewForwardRouteInfo(rep) iff rep ∈ RouteReply and
  ThereIsNoRouteInfoFor(dest(rep), RT) or
    (ThereIsRouteInfoFor(dest(rep), RT) and
      HasNewDestInfo(rep, RT))
HasNewDestInfo(rep, RT) iff                    -- entry has either
  let entry = entryFor(dest(rrep), RT)
  destSeqNum(rep) > destSeqNum(entry)          -- older destSeqNum
  or (destSeqNum(rep) = destSeqNum(entry)
      and hopCount(rep) + 1 < hopCount(entry))  -- or longer path
  or (destSeqNum(rep) = destSeqNum(entry)
      and Active(entry) = false)                -- or inactive route
  MustForward(rep) iff
    origin(rep) ≠ self and Active(entryFor(origin(rep), RT))
```

A.3.3 Route Error Components of AodvSpec

```
GenerateRouteErr =
  let UnreachEntry =
    {entry ∈ RT | LinkBreak(nextHop(entry)) and Active(entry)}
  if UnreachEntry ≠ ∅ then          -- some broken link detected
```

```
    forall entry ∈ UnreachEntry
      Active(entry) := false
      destSeqNum(entry) := destSeqNum(entry) + 1
                                        -- INCREMENT(destSeqNum(entry))
      let rerr = {(dest(e), destSeqNum(e) + 1) |
                    e ∈ UnreachEntry and precursor(e) ≠ ∅}
        forall a ∈ precursor(entry) do SEND(rerr, to a)

PROPAGATEROUTEERR =
  if Received(rerr) and rerr ∈ RouteError then
    let UnreachDest = {(d, s) ∈ rerr | forsome entry ∈ RT
                d = dest(entry) and nextHop(entry) = sender(rerr)
                and Active(entry) and destSeqNum(entry) < s}
    forall (d, s) ∈ UnreachDest
      let entry = entryFor(d, RT)
        Active(entry) := false
        destSeqNum(entry) := s
        forall a ∈ precursor(entry)  SEND(rerr', to a)
      if WaitingForRouteTo(d) then  REGENERATEROUTEREQ(d)
    CONSUME(rerr)
  where
    err' = {(d', s') ∈ UnreachDest | precursor(entryFor(d', RT)) ≠ ∅}
    REGENERATEROUTEREQ(d) = WaitingForRouteTo(d) := false
```

A.4 Relaxed Shared Memory Model (Sect. 6.2)

For the given data center system $\mathcal{D} = \bigcup_{i=1}^{k} \mathcal{D}_i$, partitioned into clusters $\mathcal{D}_i$ of data centers users interact with, two concurrent ASMs have been defined, a CLUSTERUSERVIEW$_i$ which expresses how the user is invited to perceive its interaction with the database and a CLUSTERMANAGEMENT$_i$ which describes how Cassandra manages internally the users' read and write requests.

CLUSTERUSERVIEW$_i$ = $(d, \text{DATACENTERUSERVIEW}_i)_{d \in \mathcal{D}_i}$
CLUSTERMANAGEMENT$_i$ = $(d, \text{CLUSTERDATACENTER}_i)_{d \in \mathcal{D}_i}$

A.4.1 DATACENTERUSERVIEW$_i$

```
DATACENTERUSERVIEW_i =                  -- main program of any d ∈ D_i
  ANSWERREADREQ_i
  PERFORMWRITEREQ_i

ANSWERREADREQ_i =
  if Received(read(p_i, φ), from a) then
    forall j ∈ {1, ..., q_i} -- for each fragment choose complying replicas
```

```
      choose C_{i,j} ⊆ ReplicaNodes_{i,j} with Complies(C_{i,j}, readPolicy)
        let t_max(k) =                          -- compute most recent timestamp
          max{t | p_{i,j,d',j'}(k) = (v', t) forsome v', (d', j') ∈ C_{i,j}}
                    -- collect most recent defined values in j-th fragment
        let ρ_{i,j} = {(k, v) | k ∈ Frag_{j,i} ↑ φ and v ≠ null and
                        p_{i,j,d',j'}(k) = (v, t_max(k)) forsome (d', j') ∈ C_{i,j}}
        let ρ = ⋃_{j=1}^{q_i} ρ_{i,j}           -- collect values from all fragments
          Send(ValFor(read(p_i, φ), ρ), to a)
                                    -- send current (p_i, φ)- values
    Consume(read(p_i, φ), from a)

  PerformWriteReq_i =
    if Received(write(p_i, p), from a) then
      let t_current = clock_self                -- retrieve current data center time
      forall j ∈ {1, ..., q_i} -- for each fragment choose complying replicas
        choose C_{i,j} ⊆ ReplicaNodes_{i,j} with Complies(C_{i,j}, writePolicy)
          forall (d', j') ∈ C_{i,j}             -- for each chosen d' and replica
            forall (k, v) ∈ p with h_i(k) ∈ range_j     -- for each p-value
              forall v', t with p_{i,j,d',j'}(k) = (v', t) and t < t_current
                p_{i,j,d',j'}(k) := (v, t_current)      -- update older db value
              Propagate(i, j, k, v, t_current, C_{i,j})
                              -- propagate p-value to non-chosen replicas
            if clock_{d'} < t_current then AdjustClock(d', t_current)
      Send(AckFor(write(p_i, p)), to a)
      Consume(write(p_i, p), from a)

  Propagate(i, j, k, v, t, C) =
    let b = new (Agent)
      pgm(b) :=
        forall (d', j') ∈ ReplicaNodes_{i,j} \ C
          forall v', t' with p_{i,j,d',j'}(k) = (v', t') and t' < t
            p_{i,j,d',j'}(k) := (v, t)
          if clock_{d'} < t_current then AdjustClock(d', t)
        Delete(b, Agent)[1]
```

A.4.2 ClusterDataCenter$_i$

```
  ClusterDataCenter_i =                  -- main program of any d ∈ D_i
    DelegateExternalReq_i
    ManageInternalReq_i
```

[1] Since in the ground model the propagation is formulated as happening in parallel for all involved replicas, the propagation agent performs only this one step and thus can also kill itself right away.

A.4.2.1 Request Delegation Rules

DelegateExternalReq$_i$ =
  if *Received*(*req*, **from** *a*) **then** -- any read/write *req*uest
    let $t_{current} = clock_{\mathbf{self}}$ -- retrieve current data center time
    let $c =$ **new** (*Agent*) -- create response collector
      Initialize$_i$(c, (*req*, *a*, **self**)) -- and initialize it
      Forward$_i$(*req*, c, $t_{current}$) -- delegate response
    Consume(*req*, **from** *a*)
where
  $req \in \{read(p_i, \varphi), write(p_i, p)\}$ **forsome** φ, p

Two component rules for request delegation:

Forward$_i$(r, c, t) = -- delegate response throughout the cluster
  forall $d \in \mathcal{D}_i$
    Send((r, c, t), **to** d)

Initialize$_i$(c, (r, *usr*, d)) =
  $pgm(c)$:= ManageResponseCollection$_i$ -- see below
  $ReadVal_c := \emptyset$ -- initialize set where to collect responses
  forall $d \in \mathcal{D}_i$
    forall $1 \leq j \leq q_i$
      $count_c(j, d) := 0$ -- inspected $Frag_{j,i}$-replicas at d
      $count_c(j) := 0$ -- inspected $Frag_{j,i}$-replicas
  $request_c := r$ -- record user request
  $requestor_c := usr$ -- record user
  $mediator_c := d$ -- record $home(usr)$

A.4.2.2 Internal Request Management Rules

ManageInternalReq$_i$ =
  if *Received*(*req*, c, t) **then**
    HandleLocally$_i$(*req*, c, t) -- comes with read/write versions
    Consume(*req*, c, t)

HandleLocally$_i$($read(p_i, \varphi), c, t$) =
  let $d = \mathbf{self} \in \mathcal{D}_i$ -- at the local data center d
  forall $j \in \{1, \ldots, q_i\}$ -- for each fragment
    let $G_{i,j,d}$ = -- inspect replicas at all *Alive* d-nodes
      $\{j' \mid \mathit{HoldsReplica}(j', d, j, i)$ **and** $\mathit{Alive}(j', d)\}$
    let $t_{max}(\mathbf{k})$ = -- to compute their most recent timestamp
      $\max\{t \mid p_{i,j,d,j'}(\mathbf{k}) = (\mathbf{v}', t)$ **forsome** $\mathbf{v}', j' \in G_{i,j,d}\}$
    let $\rho_{i,j,d}$ =
      $\{(\mathbf{k}, \mathbf{v}, t_{max}(\mathbf{k})) \mid \mathbf{k} \in \mathit{Frag}_{j,i} \upharpoonright \varphi$ **and** $\mathbf{v} \neq \mathit{null}$ **and**
      $p_{i,j,d,j'}(\mathbf{k}) = (\mathbf{v}, t_{\max}(\mathbf{k}))$ **forsome** $j' \in G_{i,j,d}\}$
      -- collect at d most recent defined values in j-th fragment
    let ρ_d = -- collect those values from all fragments
      $\bigcup_{j=1}^{q_i} \rho_{i,j,d}$
    let $\mathbf{x}_d$ = -- count inspected replicas
      $(|G_{i,1,d}|, \ldots, |G_{i,q_i,d}|)$
        Send($\mathit{LocalValFor}(read(p_i, \varphi), \rho_d, \mathbf{x}_d)$, **to** c)
      -- send local values to response collector

HandleLocally$_i$($write(p_i, p), c, t$) =
  let $d = \mathbf{self} \in \mathcal{D}_i$ -- at the local data center d
  forall $j \in \{1, \ldots, q_i\}$ -- for each fragment
    let $G_{i,j,d} = \{j' \mid \mathit{HoldsReplica}(j', d, j, i)$ **and** $\mathit{Alive}(j', d)\}$
      -- inspect replicas at all *Alive* d-nodes
    forall $j' \in G_{i,j,d}$ -- for each of those replicas
      forall $(\mathbf{k}, \mathbf{v}) \in p$ **with** $\mathbf{k} \in \mathit{Frag}_{j,i}$ -- for each update value in p
        if $p_{i,j,d,j'}(\mathbf{k}) = (\mathbf{v}', t')$ **with** $t' < t$ **forsome** $\mathbf{v}', t'$
          then $p_{i,j,d,j'}(\mathbf{k}) := (\mathbf{v}, t)$ -- update older values to p-value
      if $\mathit{clock}_d < t$ **then** AdjustClock(d, t)
      let $\mathbf{x} = (|G_{i,1,d}|, \ldots, |G_{i,q_i,d}|)$ -- count inspected replicas
        Send($\mathit{LocalAckFor}(write(p_i, p), \mathbf{x})$, **to** c)
      -- send local ack to response collector

A.4.2.3 Response Collection Rules

ManageResponseCollection$_i$ =
  CollectLocalReadResponses$_i$
  SendReadResponse
  CollectLocalWriteResponses$_i$
  SendWriteResponse

CollectLocalReadResponses$_i$ =
  if $\mathit{Received}(\mathit{LocalValFor}(read(p_i, \varphi), \rho, \mathbf{x}),$ **from** $d)$ **then**
    forall k if thereissome $(\mathbf{k}, \mathbf{v}, t) \in \rho$ **then** -- for each key $\mathbf{k}$
      let $(\mathbf{k}, \mathbf{v}, t) \in \rho$ -- with received local value $\mathbf{v}$
        if thereisno $(\mathbf{k}, \mathbf{v}', t') \in \mathit{ReadVal}$ -- if key $\mathbf{k}$ new for collection

```
            then INSERT((k, v, t), ReadVal) -- collect received key value
            else let (k, v′, t′) ∈ ReadVal    -- for k-value v′ in collection
              if t′ < t then                  -- with older timestamp
                DELETE((k, v′, t′), ReadVal)  -- replace old value
                INSERT((k, v, t), ReadVal)    -- by new value
        REFRESHREPLICACOUNT_i(x, d)
        CONSUME(LocalValFor(read(p_i, φ), ρ, x), from d)

    COLLECTLOCALWRITERESPONSES =
      if Received(LocalAckFor(write(p_i, p), x), from d) then
        REFRESHREPLICACOUNT_i(x, d)
        CONSUME(LocalAckFor(write(p_i, p), x), from d)

    SENDREADRESPONSE =
      if Sufficient(readPolicy)(count) and IsReadReq(request(self)) then
        let ρ = {(k, v) | (k, v, t) ∈ ReadVal forsome t}
          SEND(ValFor(request(self), ρ),      -- send the collected values
            from mediator(self), to requestor(self))
          DELETE(self, Agent)                 -- collector kills itself

    SENDWRITERESPONSE =
      if Sufficient(writePolicy)(count) and IsWriteReq(request(self)) then
        SEND(AckFor(request(self)),
          from mediator(self), to requestor(self))
        DELETE(self, Agent)
```

Auxiliary rule for replica count update:

```
    REFRESHREPLICACOUNT_i(x, d) =
      let (x_1, ..., x_{q_i}) = x
        forall j ∈ {1, ..., q_i}
          count(j, d) := count(j, d) + x_j
          count(j) := count(j) + x_j
```

References

1. Abrial, J.R.: The B-Book. Cambridge University Press, Cambridge (1996) 10, 25, 164
2. Abrial, J.R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press, Cambridge (2010) 10, 14, 36, 164
3. Abrial, J.R., Börger, E., Langmaack, H. (eds.): Methods for Semantics and Specification, *Dagstuhl Seminar Report*, vol. 117. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (1995) 239
4. Abrial, J.R., Börger, E., Langmaack, H. (eds.): Formal Methods for Industrial Applications. Specifying and Programming the Steam Boiler Control, *LNCS*, vol. 1165. Springer (1996) 239
5. Abrial, J.R., Glässer, U. (eds.): Rigorous Methods for Software Construction and Analysis (Börger Festschrift), *LNCS*, vol. 5115. Springer (2009) VI
6. Altenhofen, M., Börger, E.: Concurrent Abstract State Machines and ^{+}CAL Programs. In: A. Corradini, U. Montanari (eds.) Proc. of 19th International Workshop on Recent Trends in Algebraic Development Techniques (WADT 2008), *LNCS*, vol. 5486, pp. 1–17. Springer (2009) 174, 178, 179, 180
7. Altenhofen, M., Börger, E., Lemcke, J.: A High-Level Specification for Mediators (Virtual Providers). In: C. Bussler, A. Haller (eds.) Business Process Management Workshops: BPM 2005 International Workshops BPI, BPD, ENEI, BPRM, WSCOBPM, BPS, *LNCS*, vol. 3812, pp. 116–129. Springer (2006) 183, 187, 188
8. Altenhofen, M., Börger, E., Lemcke, J.: System and a method for mediating within a network. US Patent US7984188 B2 (2011). https://www.google.de/patents/US7984188. See also DE602005008557D1, EP1715653A1, EP1715653B1, US20060259605 182
9. Altenhofen, M., Farahbod, R.: Bârun: A Scripting Language for CoreASM. In: M. Frappier, U. Glässer, S. Khurshid, R. Laleau, S. Reeves (eds.) Proc. of 2nd International ABZ Conference ASM, Alloy, B and Z (ABZ 2010), *Theoretical Computer Science and General Issues*, vol. 5977, pp. 47–60. Springer (2010) 294
10. Altenhofen, M., Friesen, A., Lemcke, J., Börger, E.: A High-Level Specification for Virtual Providers. Int. J. Business Process Integration and Management **1**(4), 267–278 (2006) 182
11. Ameur, Y.A., Schewe, K.D. (eds.): Proc. of 4th International ABZ Conference ASM, Alloy, B, TLA, VDM, and Z (ABZ 2014), *LNCS*, vol. 8477. Springer (2014) VI, 239
12. Angluin, D., Aspnes, J., Diamadi, Z., Fischer, M.J., Peralta, R.: Computation in networks of passively mobile finite-state sensors. Distributed Computing **18**(4), 235–253 (2006) 72

E. Börger, A. Raschke, *Modeling Companion for Software Practitioners*,
https://doi.org/10.1007/978-3-662-56641-1

13. Arcaini, P., Bonfanti, S., Dausend, M., Gargantini, A., Mashkoor, A., Raschke, A., Riccobene, E., Scandurra, P., Stegmaier, M.: Unified Syntax for Abstract State Machines. In: M. Butler, K.D. Schewe, A. Mashkoor, M. Biro (eds.) Proc. of 5th International ABZ Conference ASM, Alloy, B, TLA, VDM, Z (ABZ 2016), *LNCS*, vol. 9675, pp. 231–236. Springer (2016) 295
14. Arcaini, P., Bonfanti, S., Gargantini, A., Riccobene, E.: Visual Notation and Patterns for Abstract State Machines. In: P. Milazzo, D. Varró, M. Wimmer (eds.) Proc. of Conference on Software Technologies: Applications and Foundations (STAF 2016), Collocated Workshops: DataMod, GCM, HOFM, MELO, SEMS, VeryComp, *LNCS*, vol. 9946, pp. 163–178. Springer (2016) 297
15. Arcaini, P., Gargantini, A., Riccobene, E.: Modeling and Analyzing Using ASMs: The Landing Gear System Case Study. In: F. Boniol, V. Wiels, Y.A. Ameur, K.D. Schewe (eds.) ABZ 2014: The Landing Gear Case Study, *Communications in Computer and Information Science*, vol. 433. Springer (Cham) (2014) 239
16. The Asmeta website. http://asmeta.sourceforge.net (since 2006). Consulted 01/08/2018 10, 113, 293
17. Back, R., Kurki-Suoni, R.: Decentralization of Process Nets with Centralized Control. Tech. Rep. Ser. A No. 58, Department of Computer Science at Abo Akademi, Abo (1988) 9
18. Barnett, M., Campbell, C., Schulte, W., Veanes, M.: Specification, Simulation and Testing of COM Components using Abstract State Machines. In: R. Moreno-Díaz, A. Quesada-Arencibia (eds.) Formal Methods and Tools for Computer Science (Local Proc. of Eurocast 2001), pp. 266–270 (2001) 293
19. Barnett, M., Grieskamp, W., Schulte, W., Tillmann, N., Veanes, M.: Validating use-cases with the AsmL test tool. In: Proc. of 3rd International Conference on Quality Software (QSIC'03), pp. 238–246. IEEE Computer Society (2003) 294
20. Barnett, M., Schulte, W.: Runtime Verification of .NET Contracts. J. Systems and Software **65**(3), 199–208 (2003) 293
21. Barros, A., Börger, E.: A Compositional Framework for Service Interaction Patterns and Communication Flows. In: K.K. Lau, R. Banach (eds.) Proc. of 7th International Conference on Formal Engineering Methods (ICFEM 2005), *LNCS*, vol. 3785, pp. 5–35. Springer (2005) 150, 151, 157, 159
22. Batory, D., Börger, E.: Feature Modularized Theorems in Software Product Lines. In: A. Prinz (ed.) Proc. of 14th International Worskhop on Abstract State Machines (ASM'2007) (2007) 63
23. Batory, D., Börger, E.: Modularizing Theorems for Software Product Lines: The Jbook Case Study. J. Universal Computer Science **14**(12), 2059–2082 (2008) 10, 141
24. Beauquier, D., Börger, E., Slissenko, A. (eds.): Proc. of the 12th International on Workshop on Abstract State Machines (ASM 2005) (2005) VI
25. Beierle, C., Börger, E., Durdanović, I., Glässer, U., Riccobene, E.: Refining Abstract Machine Specifications of the Steam Boiler Control to Well Documented Executable Code. In: J.R. Abrial, E. Börger, H. Langmaack (eds.) Formal Methods for Industrial Applications. Specifying and Programming the Steam-Boiler Control, no. 1165 in LNCS, pp. 62–78. Springer (1996) 239
26. Beierle, C., Finthammer, M., Potyka, N., Varghese, J., Kern-Isberner, G.: A framework for versatile knowledge and belief management operations in a probabilistic conditional logic. IFCoLog Journal of Logics and their Applications **4**(7) (2017) 9
27. Beierle, C., Kern-Isberner, G.: Modeling conditional knowledge discovery and belief revision by Abstract State Machines. In: E. Börger, A. Gargantini, E. Riccobene (eds.) Proc. of 10th International Workshop on Abstract State Machines: Advances in Theory and Practice (ASM 2003), *LNCS*, vol. 2589, pp. 186–203. Springer (2003) 9

28. Beierle, C., Kern-Isberner, G.: An ASM refinement and implementation of the Condor system using ordinal conditional functions. In: A. Prinz (ed.) Proc. of 14th International Worskhop on Abstract State Machines (ASM'2007) (2007) 9
29. Beierle, C., Kern-Isberner, G.: A high-level implementation of a system for automated reasoning with default rules (system description). In: A. Armando, P. Baumgartner, G. Dowek (eds.) Proc. of 4th International Joint Conference on Automated Reasoning (IJCAR-2008), *LNCS*, vol. 5195, pp. 147–153. Springer (2008) 9
30. Beierle, C., Kern-Isberner, G.: A verified AsmL implementation of belief revision. In: E. Börger, M. Butler, J.P. Bowen, P. Boca (eds.) Proc. of 1st International ABZ Conference ASM, B and Z (ABZ 2008), *LNCS*, vol. 5238, pp. 98–111. Springer (2008) 9
31. Beierle, C., Kern-Isberner, G.: A conceptual agent model based on a uniform approach to various belief operations. In: B. Mertsching, M. Hund, Z. Aziz (eds.) Proc. of 32nd Annual German Conference on AI: Advances in Artificial Intelligence, *Lecture Notes in Artificial Intelligence*, vol. 5803, pp. 273–280. Springer (2009) 9
32. Böhm, C., Jacopini, G.: Flow diagrams, Turing Machines, and Languages with only two Formation Rules. Commun. ACM **9**(5), 366–371 (1966) 172
33. Boniol, F., Wiels, V., Y.Ait-Ameur, Schewe, K.D. (eds.): Proc. of ABZ 2014: The Landing Gear Case Study, Case Study Track, held at the 4th International ABZ Conference ASM, Alloy, B, TLA, VDM, and Z, *Communications in Computer and Information Science*, vol. 433. Springer (2014) 239
34. Bordeleau, F., Liebel, G., Raschke, A., Stieglbauer, G., Tichy, M.: Challenges and Research Directions for Successfully Applying MBE Tools in Practice. In: Proc. of MODELS 2017 Satellite Events: MDETOOLS 2017 (2017) 318
35. Borek, M., Katkalov, K., Moebius, N., Reif, W., Schellhorn, G., Stenzel, K.: Integrating a Model-Driven Approach and Formal Verification for the Development of Secure Service Applications. In: B. Thalheim, K.D. Schewe, A. Prinz, B. Buchberger (eds.) Correct Software in Web Applications and Web Services, Texts and Monographs in Symbolic Computation, pp. 44–81. Springer (2015) 87
36. Börger, E.: A Logical Operational Semantics for Full Prolog. Part I: Selection Core and Control. In: E. Börger, H. Kleine Büning, M.M. Richter, W. Schönfeld (eds.) Proc. of 3rd Workshop on Computer Science Logic (CSL'89), *LNCS*, vol. 440, pp. 36–64. Springer (1990) 293
37. Börger, E.: A Logical Operational Semantics of Full Prolog. Part II: Built-in Predicates for Database Manipulation. In: B. Rovan (ed.) Mathematical Foundations of Computer Science, *LNCS*, vol. 452, pp. 1–14. Springer (1990) 293
38. Börger, E.: The Origins and the Development of the ASM Method for High-Level System Design and Analysis. J. Universal Computer Science **8**(1), 2–74 (2002) 14
39. Börger, E.: The ASM Refinement Method. Formal Aspects of Computing **15**, 237–257 (2003) 11
40. Börger, E.: Modeling Workflow Patterns from First Principles. In: C. Parent, K.D. Schewe, V. Storey, B. Thalheim (eds.) Proc. of 26th International Conference on Conceptual Modeling (ER 2007), *LNCS*, vol. 4801, pp. 1–20. Springer (2007) 189
41. Börger, E.: The Abstract State Machines method for high-level system design and analysis. In: P. Boca, J. Bowen, J. Siddiqi (eds.) Formal Methods: State of the Art and New Directions, pp. 79–116. Springer (2010) 17, 20, 33, 42, 43, 45, 52, 53
42. Börger, E.: Approaches to Modeling Business Processes. A Critical Analysis of BPMN, Workflow Patterns and YAWL. J. Software and Systems Modeling **11**(3), 305–318 (2012) 189, 205
43. Börger, E.: The Abstract State Machines Method for Modular Design and Analysis of Programming Languages. J. Logic and Computation **27**(2), 417–439 (2014) VI, 14, 207
44. Börger, E.: Modeling distributed algorithms by Abstract State Machines compared to Petri Nets. In: M. Butler, K.D. Schewe, A. Mashkoor, M. Biro (eds.) Proc. of

5th International ABZ Conference ASM, Alloy, B, TLA, VDM, Z (ABZ 2016), *LNCS*, vol. 9675, pp. 3–34. Springer (2016) 125
45. Börger, E., Bolognesi, T.: Remarks on Turbo ASMs for Computing Functional Equations and Recursion Schemes. In: E. Börger, A. Gargantini, E. Riccobene (eds.) Proc. of 10th International Workshop on Abstract State Machines: Advances in Theory and Practice (ASM 2003), *LNCS*, vol. 2589, pp. 218–228. Springer (2003) 171
46. Börger, E., Butler, M., Bowen, J.P., Boca, P. (eds.): Proc. of 1st International ABZ Conference ASM, B and Z (ABZ 2008), *LNCS*, vol. 5238. Springer (2008) VI
47. Börger, E., Cisternino, A., Gervasi, V.: Ambient Abstract State Machines with Applications. J. Computer and System Sciences **78**(3), 939–959 (2012) 135, 137, 140
48. Börger, E., Cisternino, A., Gervasi, V.: Modeling Web Applications Infrastructure with ASMs. Science of Computer Programming **94**(2), 69–92 (2014) 181, 182
49. Börger, E., Craig, I.: Modeling an Operating System Kernel. In: V. Diekert, K. Weicker, N. Weicker (eds.) Informatik als Dialog zwischen Theorie und Anwendung, pp. 199–216. Vieweg+Teubner (2009) 162, 168
50. Börger, E., Del Castillo, G.: A formal method for provably correct composition of a real-life processor out of basic components (The APE100 Reverse Engineering Study). In: B. Werner (ed.) Proc. of 1st IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'95), pp. 145–148. IEEE (1995) 239
51. Börger, E., Durdanović, I.: Correctness of compiling Occam to Transputer code. Computer Journal **39**(1), 52–92 (1996) 10, 204, 239
52. Börger, E., Durdanović, I., Rosenzweig, D.: Occam: Specification and Compiler Correctness. Part I: Simple Mathematical Interpreters. In: U. Montanari, E.R. Olderog (eds.) Proc. of IFIP Working Conference on Programming Concepts, Methods and Calculi (PROCOMET'94), pp. 489–508. North-Holland (1994) 204, 260
53. Börger, E., Fleischmann, A.: Abstract State Machine Nets. Closing the Gap between Business Process Models and their Implementation. In: J. Ehlers, B. Thalheim (eds.) Proc. of 7th International Conference on Subject-Oriented Business Process Management (S-BPM One 2015), S-BPM ONE '15, pp. 1:1–1:10. ACM (2015) 190, 202
54. Börger, E., Fruja, G., Gervasi, V., Stärk, R.: A high-level modular definition of the semantics of C#. Theoretical Computer Science **336**(2–3), 235–284 (2005) 4, 63, 135, 207, 318
55. Börger, E., Gargantini, A., Riccobene, E. (eds.): Proc. of 10th International Workshop on Abstract State Machines: Advances in Theory and Practice (ASM 2003), *LNCS*, vol. 2589. Springer (2003) VI
56. Börger, E., Gargantini, A., Riccobene, E.: Abstract State Machines. A Method for System Specification and Analysis. In: M. Frappier, H. Habrias (eds.) Software Specification Methods: An Overview Using a Case Study, pp. 103–119. HERMES Sc. Publ. (2006) 205, 239
57. Börger, E., Glässer, U., Müller, W.: The Semantics of Behavioral VHDL'93 Descriptions. In: J. Mermet (ed.) Proc. of European Design Automation Conference (EURO-DAC'94) with EURO-VHDL'94, pp. 500–505. IEEE Computer Society Press (1994) 4
58. Börger, E., Glässer, U., Müller, W.: Formal Definition of an Abstract VHDL'93 Simulator by EA-Machines. In: C. Delgado Kloos, P.T. Breuer (eds.) Formal Semantics for VHDL, pp. 107–139. Springer (1995) 4
59. Börger, E., Gotzhein, R.: The Light Control Case Study. J. Universal Computer Science **6**(7), 580–585 (2000) 239
60. Börger, E., Hörger, B., Parnas, D.L., Rombach, D. (eds.): Requirements Capture, Documentation, and Validation, *Dagstuhl Seminar Report*, vol. 241. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (1999) 239

61. Börger, E., Joannou, P., Parnas, D.L. (eds.): Practical Methods for Code Documentation and Inspection, *Dagstuhl Seminar Report*, vol. 178. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (1997) 239
62. Börger, E., Leuschel, M.: A compact encoding of sequential ASMs in Event-B. In: M. Butler, K.D. Schewe, A. Mashkoor, M. Biro (eds.) Proc. of 5th International ABZ Conference ASM, Alloy, B, TLA, VDM, Z (ABZ 2016), *LNCS*, vol. 9675, pp. 119–134. Springer (2016) 10
63. Börger, E., Mazzanti, S.: A Practical Method for Rigorously Controllable Hardware Design. In: J.P. Bowen, M.B. Hinchey, D. Till (eds.) ZUM'97: The Z Formal Specification Notation, *LNCS*, vol. 1212, pp. 151–187. Springer (1997) 239
64. Börger, E., Mearelli, L.: Integrating ASMs into the Software Development Life Cycle. J. Universal Computer Science **3**(5), 603–665 (1997) 239, 297
65. Börger, E., Päppinghaus, P., Schmid, J.: Report on a Practical Application of ASMs in Software Design. In: Y. Gurevich, P. Kutter, M. Odersky, L. Thiele (eds.) Proc. of International Workshop on Abstract State Machines: Theory and Applications (ASM 2000), *LNCS*, vol. 1912, pp. 361–366. Springer (2000) 164, 293
66. Börger, E., Prinz, A.: Special ASM Issue with Selected Papers from ASM 2007. J. Universal Computer Science **14**(12) (2008) VI
67. Börger, E., Riccobene, E., Schmid, J.: Capturing Requirements by Abstract State Machines: The Light Control Case Study. J. Universal Computer Science **6**(7), 597–620 (2000) 239, 293
68. Börger, E., Rosenzweig, D.: A Mathematical Definition of Full Prolog. Science of Computer Programming **24**, 249–286 (1995) 4, 31
69. Börger, E., Rosenzweig, D.: The WAM – Definition and Compiler Correctness. In: C. Beierle, L. Plümer (eds.) Logic Programming: Formal Methods and Practical Applications, *Studies in Computer Science and Artificial Intelligence*, vol. 11, chap. 2, pp. 20–90. North-Holland (1995) 4, 10, 31
70. Börger, E., Schewe, K.D.: Concurrent Abstract State Machines. Acta Informatica **53**(5), 469–492 (2016) 73, 132, 225, 272
71. Börger, E., Schewe, K.D.: Communication in Abstract State Machines. J. Universal Computer Science **23**(2), 129–145 (2017) 115, 126
72. Börger, E., Schewe, K.D., Wang, Q.: Serialisable Multi-Level Transaction Control:A Specification and Verification. Science of Computer Programming **131**, 42–85 (2016) 82, 194
73. Börger, E., Schmid, J.: Composition and Submachine Concepts for Sequential ASMs. In: P. Clote, H. Schwichtenberg (eds.) Proc. of 14th International Workshop on Computer Science Logic (CSL 2000), *LNCS*, vol. 1862, pp. 41–60. Springer (2000) 171
74. Börger, E., Slissenko, A. (eds.): Special ASM Issue of Fundamenta Informaticae with Selected Papers from ASM 2005. IOS Press (Amsterdam) (2007) VI
75. Börger, E., Sörensen, O.: BPMN Core Modeling Concepts: Inheritance-Based Execution Semantics. In: D. Embley, B. Thalheim (eds.) Handbook of Conceptual Modelling. Theory, Practice, and Research Challenges, pp. 287–332. Springer (2011) 141
76. Börger, E., Stärk, R.F.: Abstract State Machines. A Method for High-Level System Design and Analysis. Springer (2003) VI, VIII, XI, 13, 25, 33, 75, 90, 91, 120, 148, 154, 166, 172, 243, 245, 247, 248, 249, 251, 252, 254, 293, 297
77. Börger, E., Stärk, R.F.: Exploiting Abstraction for Specification Reuse. The Java/C# Case Study. In: F.S. de Boer, M.M. Bonsangue, S. Graf, W.P. de Roever (eds.) Formal Methods for Components and Objects: Second International Symposium (FMCO 2003), *LNCS*, vol. 3188, pp. 42–76. Springer (2004) 318
78. Börger, E., Thalheim, B.: A Method for Verifiable and Validatable Business Process Modeling. In: E. Börger, A. Cisternino (eds.) Advances in Software Engineering, *LNCS*, vol. 5316, pp. 59–115. Springer (2008) 205

79. Börger, E., Zenzaro, S.: Business Process Modeling for Reuse and Change via Component-Based Decomposition and ASM Refinement. In: J. Ehlers, B. Thalheim (eds.) Proc. of 7th International Conference on Subject-Oriented Business Process Management (S-BPM One 2015), S-BPM One. ACM (2015) 65, 66, 68, 70, 71, 75, 83, 84, 86
80. Bosa, K.: Formal modeling of mobile computing systems based on ambient Abstract State Machines. In: K.D. Schewe, B. Thalheim (eds.) Proc. of 5th International Workshop Semantics in Data and Knowledge Bases (SDKB 2011), *LNCS*, vol. 7693, pp. 18–49. Springer (2013) 135
81. Burgard, W., Cremers, A.B., Fox, D., Heidelbach, M., Kappel, A.M., Lüttringhaus-Kappel, S.: Knowledge-enhanced CO-monitoring in Coal Mines. In: Proc. of International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA-AIE), pp. 511–521 (1996) 293
82. Butler, M., Schewe, K.D., Mashkoor, A., Biro, M. (eds.): Proc. of 5th International ABZ Conference ASM, Alloy, B, TLA, VDM, and Z (ABZ 2016), *LNCS*, vol. 9675. Springer (2016) VI
83. Cardelli, L., Gordon, A.: Mobile Ambients. Theoretical Computer Science **240**(1), 177–213 (2000) 135
84. The CoreASM Project. http://www.coreasm.org and https://github.com/coreasm/ (since 2005). Consulted 01/08/2018 10, 96, 103, 113, 293
85. Craig, I., Börger, E.: Synchronous Message Passing and Semaphores: An Equivalence Proof. In: M. Frappier, U. Glässer, S. Khurshid, R. Laleau, S. Reeves (eds.) Proc. of 2nd International ABZ Conference ASM, Alloy, B and Z (ABZ 2010), *LNCS*, vol. 5977, pp. 20–33. Springer (2010) 150, 159, 164
86. da Cruz, F.: Kermit: A File Transfer Protocol. Digital Press (1986). See also the Kermit Web site http://www.columbia.edu/kermit 4
87. Dausend, M., Stegmaier, M., Raschke, A.: Debugging Abstract State Machine Specifications: An Extension of CoreASM. In: F. Mazzatini, G. Trentanni (eds.) Proc. of Posters & Tool demos Session, held at 9th International Conference on Integrated Formal Methods and 3rd International ABZ Conference ASM, Alloy, B, VDM, Z (iFM 2012 & ABZ 2012), pp. 21–25 (2012) 286, 292
88. Del Castillo, G.: The ASM Workbench. A Tool Environment for Computer-Aided Analysis and Validation of Abstract State Machine Models. Ph.D. thesis, Universität Paderborn (2001) 293
89. Del Castillo, G., Winter, K.: Model Checking Support for the ASM High-Level Language. In: S. Graf, M. Schwartzbach (eds.) Proc. of the 6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2000), *LNCS*, vol. 1785, pp. 331–346. Springer (2000) 293
90. Denvir, T., Oliveira, J., Plat, N.: The Cash-Point (ATM) 'Problem'. Formal Aspects of Computing **12**(4), 211–215 (2000) 64
91. Derrick, J., Fitzgerald, J.S., Gnesi, S., Khurshid, S., Leuschel, M., Reeves, S., Riccobene, E. (eds.): Proc. of 3rd International ABZ Conference ASM, Alloy, B, VDM and Z (ABZ 2012), *LNCS*, vol. 7316. Springer, Pisa (Italy) (2012) VI
92. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik **1**(1), 269–271 (1959) 75
93. Dijkstra, E.W.: Guarded commands, non-determinacy and formal derivation of programs. Commun. ACM **18**(8), 453–457 (1975). Also available as EWD 472 9
94. Dijkstra, E.W.: Shmuel Safra's version of termination detection. In: M. Broy, R. Steinbrüggen (eds.) Proc. of NATO Advanced Study Institute on Calculational System Design, pp. 297–301 (1999) 106, 109, 111, 114
95. Dijkstra, E.W., J.Feijen, W.H., van Gasteren, A.J.M.: Derivation of a termination detection algorithm for distributed computations. Information Processing Letters **16**(5), 217–219 (1983). Known also as EWD 840 103, 106, 108, 109, 111, 112, 113, 115

96. Dijkstra, E.W., Scholten, C.S.: Termination detection for diffusing computations. Information Processing Letters **11**(1), 1–4 (1980). Known also as EWD 687a 121, 122, 124
97. Dmitirev, S.: Language Oriented Programming: The Next Programming Paradigm. http://www.jetbrains.com (2004) 12
98. Dold, A.: A Formal Representation of Abstract State Machines Using PVS. Verifix Technical Report Ulm/6.2, Universität Ulm (1998) 10, 293
99. Dold, A., Gaul, T., Zimmermann, W.: Mechanized Verification of Compiler Back-Ends. In: T. Margaria, B. Steffen (eds.) Proc. of the International Workshop on Software Tools for Technology Transfer (STTT'98) (1998) 293
100. Dumas, M., Rosa, M.L., Mendling, J., Reijers, H.: Fundamentals of Business Process Management, chap. Introduction to Business Process Management, pp. 1–31. Springer (2013) 205
101. E. Börger O. Sörensen, B.T.: On defining the behavior of OR-joins in business process models. J. Universal Computer Science **15**(1), 3–32 (2009) 203
102. Eschbach, R.: A Termination Detection Algorithm: Specification and Verification. In: J. Wing, J.C.P. Woodcock, J. Davies (eds.) Proc. of International Conference on Formal Methods in the Development of Computing Systems (FM'99), Vol. II, *LNCS*, vol. 1709, pp. 1720–1737. Springer (1999) 106, 109, 111, 113, 114
103. Eschbach, R., Glässer, U., Gotzhein, R., v. Löwis, M., Prinz, A.: Formal Definition of SDL-2000 – Compiling and Running SDL Specifications as ASM Models. J. Universal Computer Science **7**(11), 1025–1050 (2001) 4, 293, 297
104. Farahbod, R.: CoreASM: an extensible modeling framework & tool environment for high-level design and analysis of distributed systems. Ph.D. thesis, Simon Fraser University (2009) 252, 253, 272, 285
105. Farahbod, R.: Design and Specification of the CoreASM Execution Engine and Plugins. https://github.com/CoreASM/coreasm.core/wiki/Documentation (2010) 252, 253, 254, 255, 256, 259, 263, 265, 274, 284
106. Farahbod, R., Dausend, M.: CoreASM Language User Manual. https://github.com/CoreASM/coreasm.core/wiki/Documentation (2016) 252, 278
107. Farahbod, R., Gervasi, V., Glässer, U.: CoreASM: An Extensible ASM Execution Engine. Fundamenta Informaticae XXI **77**(1-2), 71–103 (2007) 253
108. Fehnker, A., van Glabbeek, R., Hoefner, P., McIver, A., Portmann, M., Tan, W.L.: A process algebra for wireless mesh networks used for modelling, verifying and analysing AODV. Tech. Rep. 5513, NICTA, Brisbane (Australia) (2013) 208
109. Fleischmann, A., Schmidt, W., Stary, C., Obermeier, S., Börger, E.: Subject-Oriented Business Process Management. Springer (2012). http://www.springer.com/978-3-642-32391-1 (Open Access Book) 90, 117, 157, 182, 194, 196, 204
110. Fleischmann, A., Stary, C.: Whom to talk to? A stakeholder perspective on business process development. Universal Access in the Information Society **11**(2), 125–150 (2012) 90, 181
111. Foundation, A.S.: Apache Cassandra 2.0 – Documentation. http://cassandra.apache.org/ (2016). Consulted 01/08/2018 224, 225
112. Foundations of Software Engineering Group, Microsoft Research: AsmL. Web pages at https://www.microsoft.com/en-us/research/project/asml-abstract-state-machine-language/, consulted 01/08/2018 (2001) 293
113. Franklin, R.: On an Improved Algorithm for Decentralized Extrema Finding in Circular Configurations of Processors. Commun. ACM **25**(5), 336–337 (1982) 96, 100
114. Frappier, M., Glässer, U., Khurshid, S., Laleau, R., Reeves, S. (eds.): Proc. of 2nd International ABZ Conference ASM, Alloy, B, and Z (ABZ 2010), *LNCS*, vol. 5977. Springer (2010) VI
115. Frappier, M., Habrias, H. (eds.): Software Specification Methods: An Overview Using a Case Study. ISTE (2006) 205, 239

116. Friesen, A., Börger, E.: A High-Level Specification for Semantic Web Service Discovery Services. In: O. Pastor, O.Corcho, P. Sampaio, G.J. Houben, D. Schwabe, A. Wombacher (eds.) Workshop Proc. of the 6th International Conference on Web Engeneering (ICWE'06). Joint Workshop on Web Services Modeling and Implementation using Sound Web Engineering Practices and Methods, Architectures and Technologies for e-Service Engineering (SMIWEP-MATeS'06). ACM (2006) 182
117. Fruja, N.G.: The correctness of the definite assignment analysis in C#. Journal of Object Technology **3**(9), 29–52 (2004) 135
118. Fruja, N.G.: Type Safety of C# and .NET CLR. Ph.D. thesis, ETH Zürich (2006) 4, 207
119. Fruja, N.G.: Towards proving type safety of .NET CIL. Science of Computer Programming **72**(3), 176–219 (2008) 135
120. Fruja, N.G., Börger, E.: Modeling the .NET CLR Exception Handling Mechanism for a Mathematical Analysis. J. of Object Technology **5**(3), 5–34 (2006) 135
121. Fruja, N.G., Stärk, R.F.: The hidden computation steps of Turbo Abstract State Machines. In: E. Börger, A. Gargantini, E. Riccobene (eds.) Proc. of 10th International Workshop on Abstract State Machines: Advances in Theory and Practice (ASM 2003), *LNCS*, vol. 2589, pp. 244–262. Springer (2003) 172
122. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns. Addison-Wesley (1994) 137, 145, 146, 147, 148, 149
123. Gardner, M.: Mathematical games. Scientific American **223**, 120–123 (1970) 63
124. Gargantini, A., Riccobene, E.: Encoding Abstract State Machines in PVS. In: Y. Gurevich, P. Kutter, M. Odersky, L. Thiele (eds.) Proc. of International Workshop on Abstract State Machines: Theory and Applications (ASM 2000), *LNCS*, vol. 1912, pp. 303–322. Springer (2000) 10, 239
125. Gervasi, V.: An ASM Model of Concurrency in a Web Browser. In: J.D. et al. (ed.) Proc. of 3rd International ABZ Conference ASM, Alloy, B, VDM, and Z (ABZ 2012), *LNCS*, vol. 7316, pp. 79–93. Springer (2012) 181
126. Gervasi, V., Riccobene, E.: From English to ASM: On the process of deriving a formal specification from a natural language one. Dagstuhl Report **3**(9), 85–90 (2013) 103, 111, 112, 113
127. Ghezzi, G., Jazayeri, M., Mandrioli, D.: Fundamentals of Software Engineering. Prentice-Hall (1991) 33
128. Giese, M., Kempe, D., Schönegge, A.: KIV zur Verifikation von ASM-Spezifikationen am Beispiel der DLX-Pipelining Architektur. Tech. Rep. 16/97, Universität Karlsruhe, Fakultät für Informatik (1997) 10, 239
129. Gjøsæter, T., Prinz, A., Scheidgen, M.: Meta-model or Grammar? Methods and Tools for the Formal Definition of Languages. In: Nordic Workshop on Model Driven Engineering (NW-MoDE 2008), pp. 67–82. University of Iceland Press (2008) 298
130. Glässer, U., Gurevich, Y., Veanes, M.: Abstract communication model for distributed systems. IEEE Transactions on Software Engineering **30**(7), 458–472 (2004) 158
131. Goerigk, W., Dold, A., Gaul, T., Goos, G., Heberle, A., von Henke, F.W., Hoffmann, U., Langmaack, H., Pfeifer, H., Ruess, H., Zimmermann, W.: Compiler Correctness and Implementation Verification: The Verifix Approach. In: P. Fritzson (ed.) Proceedings of the Poster Session of International Conference on Compiler Construction (CC '96), pp. 65 – 73. IDA Technical Report LiTH-IDA-R-96-12, Linköping (1996) 293
132. Goncalves, R.C., Batory, D., Sobral, J.L., Riché, T.L.: From Software Extensions to Product Lines of Dataflow Programs. Software & Systems Modeling **16**(4), 929–947 (2017) 13
133. González-Pérez, C.A.: Metamodelling for Software Engineering. J. Wiley (2008) 298

134. Granger, C.: Coding is not the new literacy. http://www.chris-granger.com/2015/01/26/coding-is-not-the-new-literacy/ (2015). Consulted 01/08/2018 VII, 7
135. Grune, D., Reeuwijk, K.v., Bal, H., Jacobs, C., Langendoen, K.: Modern Compiler Design, 2nd edn. Springer (2012) 264, 265
136. Gurevich, Y.: Evolving algebras 1993: Lipari Guide. In: E. Börger (ed.) Specification and Validation Methods, pp. 9–36. Oxford University Press (1995) 132
137. Gurevich, Y., Tillmann, N.: Partial Updates: Exploration. J. Universal Computer Science **7**(11), 918–952 (2001) 260
138. Gurevich, Y., Tillmann, N.: Partial Updates. Theoretical Computer Science **336**(2-3), 311–342 (2005) 260
139. Haneberg, D., Grandy, H., Reif, W., Schellhorn, G.: Verifying Smart Card Applications: An ASM Approach. In: Proc. of International Conference on Integrated Formal Methods (iFM 2007), *LNCS*, vol. 4591. Springer (2007) 91, 293
140. Haneberg, D., Junker, M., Schellhorn, G., Reif, W., Ernst, G.: Simulating a Flash File System with CoreASM and Eclipse. In: H.U. Hei, P. Pepper, H. Schlingloff, J. Schneider (eds.) INFORMATIK 2011. Informatik schafft Communities, *LNI*, vol. 192. Bonner Köllen Verlag (2011) 91, 293
141. Haneberg, D., Moebius, N., Reif, W., Schellhorn, G., Stenzel, K.: Mondex: Engineering a Provable Secure Electronic Purse. International Journal of Software and Informatics **5**(1), 159–184 (2011) 10, 91
142. Haneberg, D., Schellhorn, G., Grandy, H., Reif, W.: Verification of Mondex electronic purses with KIV: from transactions to a security protocol. Formal Aspects of Computing **20**(1), 41–59 (2008) 10, 91
143. Heath, F., Boaz, D., Gupta, M., Vaculín, R., Sun, Y., Hull, R., Limonad, L.: Barcelona: A design and runtime environment for declarative artifact-centric BPM. In: S. Basu, C. Pautasso, L. Zhang, X. Fu (eds.) Proc. of 11th International Conference on Service-Oriented Computing (ICSOC 2013), *LNCS*, vol. 8274, pp. 705–709 (2013) 193
144. Hohpe, G., Woolf, B.: Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley (2003) 182
145. Hommel, G.: Vergleich verschiedener Spezifikationsverfahren am Beispiel einer Paketverteilanlage. Tech. Rep. KfK-PDV 186, Kernforschungszentrum Karlsruhe (1980) 52, 55, 56, 59, 61, 62
146. Huggins, J.: Kermit: Specification and Verification. In: E. Börger (ed.) Specification and Validation Methods, pp. 247–293. Oxford University Press (1995) 4
147. Huggins, J., Van Campenhout, D.: Specification and Verification of Pipelining in the ARM2 RISC Microprocessor. ACM Trans. Des. Autom. of Electron. Syst. **3**(4), 563–580 (1998) 239
148. Hull, R., Damaggio, E., Fournier, F., Gupta, M., Heath, F., Hobson, S., Linehan, M., Maradugu, S., Nigam, A., Sukaviriya, P., Vaculin, R.: Introducing the Guard-Stage-Milestone Approach for Specifying Business Entity Lifecycles. In: M. Bravetti, T. Bultan (eds.) Proc. of 7th International Workshop on Web Services and Formal Methods (WS-FM 2010), *LNCS*, vol. 6551, pp. 1–24. Springer (2011) 90, 181, 193
149. Hutton, G., Meijer, E.: Monadic Parsing in Haskell. Journal of Functional Programming **8**(4), 437–444 (1998) 265
150. IEEE: IEEE Standard VHDL Language Reference Manual (1993). IEEE Std 1076-1993 4
151. ISO: Prolog-Part 1: General Core. ISO Standard Information Technology–Programming Languages ISO/IEC 13211-1, ISO/ICE (1995) 4
152. ISO/IEC: Vienna Development Method — Specification Language — Part 1: Base language (1996). Information Technology ISO/IEC 13817-1 Standard 164

153. ISO/IEC: Z Formal Specification Notation âĂŤ Syntax, Type System and Semantics. http://www.iso.org/iso/catalogue_detail?csnumber=21573 (2002). Information Technology ISO/IEC 13568 Standard 164
154. ITU-T: SDL Formal Semantics Definition. ITU-T Recommendation Z.100 Annex F, International Telecommunication Union (2000). http://www.sdl-forum.org 4
155. Jackson, M.: Problem Frames: Analyzing and Structuring Software Development Problems. Addison-Wesley (2001) 14, 17, 42, 47, 48
156. Jones, C., Hayes, I.J., Jackson, M.A.: Specifying systems that connect to the physical world. Tech. Rep. CS-TR-964, University of Newcastle upon Tyne (2006) 46
157. Kappel, A.M.: Implementation of Dynamic Algebras with an Application to Prolog. Diploma thesis, Universität Dortmund (1990) 293
158. Kappel, A.M.: Executable Specifications Based on Dynamic Algebras. In: A. Voronkov (ed.) Proc. of 4th International Conference on Logic Programming and Automated Reasoning (LPAR'93), *LNAI*, vol. 698, pp. 229–240. Springer (1993) 293
159. Karg, S., Raschke, A., Tichy, M., Liebel, G.: Model-driven Software Engineering in the openETCS Project: Project Experiences and Lessons Learned. In: Proc. of the 19th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS'16), MODELS '16, pp. 238–248. ACM, New York, NY, USA (2016) 318
160. Kern-Isberner, G.: Conditionals in Nonmonotonic Reasoning and Belief Revision, *LNAI*, vol. 2087. Springer (2001) 9
161. The KIV System. http://www.isse.uni-augsburg.de/en/software/kiv/ (since 2013). Consulted 01/08/2018 32, 91
162. Knuth, D.: Structured Programming with goto Statements. ACM Computing Surveys **6**(4), 261–301 (1974) 35, 67
163. Knuth, D.E.: Literate Programming. Comput. J. **27**(2), 97–111 (1984) 277
164. Kohlmeyer, J.: Eine formale Semantik für die Verknüpfung von Verhaltensbeschreibungen in der UML 2. Ph.D. thesis, Universität Ulm (2009) 192
165. Kohlmeyer, J., Guttmann, W.: Unifying the Semantics of UML 2 State, Activity and Interaction Diagrams. In: A. Pnueli, I. Virbitskaite, A. Voronkov (eds.) Proc. of the 7th International Andrei Ershov Memorial Conference on Perspectives of Systems Informatics (PSI 2009), *LNCS*, vol. 5947, pp. 206–217. Springer (2010) 192
166. Kossak, F., Illibauer, C., Geist, V., Kubovy, J., Natschläger, C., Ziebermayr, T., Kopetzky, T., Freudenthaler, B., Schewe, K.D.: A Rigorous Semantics fo BPMN 2.0 Process Diagrams. Springer (2014) 4, 205
167. Kossak, F., Illibauer, C., Geist, V., Natschläger, C., Ziebermayr, T., Freudenthaler, B., Kopetzky, T., Schewe, K.D.: Hagenberg Business Process Modelling Method. Springer (2016) 205
168. Lamport, L.: A New Solution of Dijkstra's Concurrent Programming Problem. Commun. ACM **17**(8), 453–455 (1974) 133
169. Lamport, L.: Time, Clocks, and the Ordering of Events in a Distributed System. Commun. ACM **21**(7), 558–565 (1978) 117, 127, 156, 228
170. Lamport, L.: How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs. IEEE Trans. Computers **28**(9), 690–691 (1979) 225
171. Lamport, L.: On Interprocess Communication. Part I: Basic Formalism. Part II: Algorithms. Distributed Computing **1**(2), 77–101 (1986) 133
172. Lamport, L.: A Fast Mutual Exclusion Algorithm. ACM Transactions of Computer Systems **5**(1), 1–11 (1987) 180
173. Lamport, L.: Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers. Addison-Wesley (2003) 164
174. Lamport, L.: A +CAL User's Manual.P-Syntax Version. http://lamport.azurewebsites.net/tla/p-manual.pdf (2017) 176, 177, 178

175. Lamport, L.: The +CAL algorithm Language. http://lamport.azurewebsites.net/pubs/pluscal.pdf (2017) 172, 174, 176
176. Lampson, B.W.: Principles of Computer Systems. MIT Lecture Notes 6.826, http://bwlampson.site/48-POCScourse/48-POCS2006.pdf (2006). Consulted 01/07/2018 165
177. Langmaack, H.: An ALGOL-view on TURBO ASM. In: W. Zimmermann, B. Thalheim (eds.) Proc. of 11th International Workshop on Abstract State Machines: Advances in Theory and Practice (ASM 2004), *LNCS*, vol. 3052, pp. 20–37. Springer (2004) 172
178. Leibniz, G.W.: Dialogus de connexione inter res et verba. G. W. Leibniz: Philosophische Schriften (1677). Edited by Leibniz-Forschungsstelle der Universität Münster, Vol. 4 A, n. 8. Akademie Verlag 1999 7
179. Leitz, M.: Definition of the formal semantics of Control State Diagrams and implementation of a graphical editor. Master's thesis, Ulm University, Ulm (2018) 297, 299
180. Lerchner, H.: Open S-BPM Workflow Engine. http://www.i2pm.net/interest-groups/open-s-bpm, consulted 01/08/2018 (2013) 182, 194
181. Lerchner, H., Stary, C.: An Open S-BPM Runtime Environment Based on Abstract State Machines (CBI 2014). In: Proc. of 16th IEEE Conference on Business Informatics 2014, vol. 1, pp. 54–61 (2014). See http://www.i2pm.net/interest-groups/open-s-bpm/sub-projects/open-s-bpm-workflow-engine 182, 194
182. Leveson, N.G.: Engineering a Safer World: Systems Thinking Applied to Safety. Engineering Systems. MIT Press (2012) 5, 7, 318
183. Lewerentz, C., Lindner, T.: Formal Development of Reactive Systems. Case Study "Production Cell", *LNCS*, vol. 891. Springer (1995) 239
184. Lynch, N.: Distributed Algorithms. Morgan Kaufmann (1996) 117, 119, 125, 126
185. Mearelli, L.: Refining an ASM Specification of the Production Cell to C++ Code. J. Universal Computer Science **3**(5), 666–688 (1997) 164, 239
186. Memon, M.: Specification Language Design Concepts: Aggregation and Extensibility in CoreASM. Master thesis, Simon Fraser University (2006) 262
187. Metasonic: Metasonic Tool Suite (2017). https://www.metasonic.de/en/s-bpm, consulted 01/08/2018 182, 194
188. Monteverdi, C.: Quinto libro di madrigali a 5 voci: prefazione. Claudio Monteverdi (1605) V
189. Monteverdi, G.C. (ed.): Scherzi Musicali di Claudio Monteverdi, chap. Dichiaratione della lettera stampata nel quinto libro dei suoi madrigali (1607) V
190. Moore, E.F.: The Shortest Path Through a Maze. In: Proc. Int. Sympos. on Theory of Switching, *The Annals of the Computation Laboratory of Harvard University*, vol. 30. II. Harvard University Press (1959) 75
191. Mukala, P., Cerone, A., Turini, F.: An abstract state machine (ASM) representation of learning process in FLOSS communities. In: C. Canal, A. Idani (eds.) Software Engineering and Formal Methods. Proc. of Colocated Workshops HOFM, SAFOME, OpenCert, MoKMaSD, WS-FMDS (SEFM 2014), *LNCS*, vol. 8938, pp. 227–242. Springer (2014) 297
192. Naur, P.: Programming as theory building. Microprocessing and Microprogramming **15**(5), 253–261 (1985) 5
193. Oaks, S., Wong, H.: Java Threads, third edn. O'Reilly (2004). See also https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ThreadPoolExecutor.html 140, 141
194. Ober, I.: More Meaningful UML Models. In: Proc. of 37th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS-Pacific 2000), pp. 146–157. IEEE Computer Society Press (2000) 293
195. Object Management Group (OMG): UML 2.0 Superstructure Specification. http://www.omg.org/cgi-bin/doc?formal/05-07-04 (2004) 192

196. Object Management Group (OMG): Business Process Model and Notation (BPMN). Version 2.0. http://www.omg.org/spec/BPMN/2.0 (2011). Formal/2011-01-03 4, 141, 185, 189, 192, 205
197. Object Management Group (OMG): Case Management Model and Notation (CMMN). Version 1.0 Beta 1. http://www.omg.org/spec/CMMN/1.0/Beta1 (2013) 193
198. Orwell, G.: Politics and the English Language. Horizon **13**(76), 252–265 (1946) X
199. Ousterhout, J.: Tcl and the Tk Toolkit. Addison-Wesley (1994). See also the TCL developer site http://www.tcl.tk 139
200. Parnas, D., Lawford, M.: The Role of Inspection in Software Quality Assurance. IEEE Transactions on Software engineering **29**(8), 674–676 (2003) 12
201. Perkins, C., Belding-Royer, E., Das, S.: Ad hoc On-Demand Distance Vector (AODV) Routing. RFC 3561, Copyright (C) The Internet Society, Network Working Group (2003). URL http://tools.ietf.org/html/rfc3561 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 221, 222, 224
202. Perkins, C., Royer, E.: Ad-hoc On-Demand Distance Vector Routing. In: Proc. of the 2nd IEEE Workshop on Mobile Computer Systems and Applications (WMCSA'99), pp. 90–100 (1999) 208, 215, 218, 221
203. Petre, L., Sekerinski, E.: From Action Systems to Distributed Systems: The Refinement Approach. Chapman and Hall/CRC (2016) 9
204. Popper, K.: Logik der Forschung. Springer (1935) 8
205. Riccobene, E., Reeves, S.: Selected and extended papers from ABZ 2012. Special Issue of *Science of Computer Programming* **94**(2) (2014) VI
206. Riccobene, E., Scandurra, P.: A formal framework for service modeling and prototyping. Formal Aspects of Computing **26**(6), 1077–1113 (2014) 156
207. Rothstein, E., Schreckling, D.: Execution Framework for Interaction Computing (2016). Deliverable D5.2. Available from http://www.biomicsproject.eu/file-repository/category/BIOMICS-d52-FinalVersion.pdf, consulted 01/08/2018 269, 293
208. Sarstedt, S.: Semantic Foundation and Tool Support for Model-Driven Development with UML 2 Activity Diagrams. Ph.D. thesis, Universität Ulm (2006) 192, 297
209. Sarstedt, S., Guttmann, W.: An ASM Semantics of Token Flow in UML 2 Activity Diagrams. In: I. Virbitskaite, A. Voronkov (eds.) Proc. of the 6th International Andrei Ershov Memorial Conference on Perspectives of System Informatics (PSI 2006), *LNCS*, vol. 4378, pp. 349–362. Springer (2007) 192
210. Sasaki, H.: A Formal Semantics for Verilog-VHDL Simulation Interoperability by Abstract State Machines. In: Proc. of IEEE Conference on Design, Automation and Test in Europe (DATE'99), pp. 353–357 (1999) 4
211. Sasaki, H., Mizushima, K., Sasaki, T.: Semantic Validation of VHDL-AMS by an Abstract State Machine. In: Proc. of the IEEE/VIUF International Workshop on Behavioral Modeling and Simulation (BMAS'97), pp. 61–68 (1997) 4
212. Schellhorn, G.: Verifikation abstrakter Zustandsmaschinen. Ph.D. thesis, Universität Ulm (1999) 32, 91
213. Schellhorn, G.: Verification of ASM Refinements Using Generalized Forward Simulation. J. Universal Computer Science **7**(11), 952–979 (2001) 32, 91
214. Schellhorn, G.: ASM Refinement and Generalizations of Forward Simulation in Data Refinement: A Comparison. Theoretical Computer Science **336**(2-3), 403–435 (2005) 32, 91
215. Schellhorn, G.: ASM Refinement Preserving Invariants. J. UCS **14**(12), 1929–1948 (2008) 32, 91
216. Schellhorn, G.: Completeness of ASM Refinement. Electr. Notes Theor. Comput. Sci. **214**, 25–49 (2008) 32, 91
217. Schellhorn, G.: Completeness of fair ASM refinement. Sci. Comput. Program. **76**(9), 756–773 (2011) 32, 91

218. Schellhorn, G., Ahrendt, W.: The WAM Case Study: Verifying Compiler Correctness for Prolog with KIV. In: W. Bibel, P.H. Schmitt (eds.) Automated Deduction – A Basis for Applications, vol. 10, pp. 165–194. Springer (1998) 10, 91, 293
219. Schellhorn, G., Grandy, H., Haneberg, D., Moebius, N., Reif, W.: A Systematic Verification Approach for Mondex Electronic Purses Using ASMs. In: J.R. Abrial, U. Glässer (eds.) Rigorous Methods for Software Construction and Analysis: Essays Dedicated to Egon Börger on the Occasion of His 60th Birthday, *LNCS*, vol. 5115, pp. 93–110. Springer (2009) 10, 293
220. Schewe, K.D., Ferrarotti, F., Tec, L., Wang, Q., An, W.: Evolving Concurrent Systems: Behavioural Theory and Logic. In: Proc. of the Australasian Computer Science Week Multiconference (ACSW 2017), pp. 77:1–77–10. ACM (2017) 165
221. Schewe, K.D., Prinz, A., Börger, E.: Concurrent computing with Shared Memory Models. submitted for publication (2017) 224, 231, 232, 238
222. Schmid, J.: Executing ASM Specifications with AsmGofer. Web pages at https://tydo.eu/AsmGofer (1999). 293
223. Schmid, J.: Compiling Abstract State Machines to C++. J. Universal Computer Science **7**(11), 1069–1087 (2001) 239, 293
224. Schmid, J.: Refinement and Implementation Techniques for Abstract State Machines. Ph.D. thesis, University of Ulm (2002) 293
225. Schönegge, A.: Extending Dynamic Logic for Reasoning about Evolving Algebras. Technical Report 49/95, Universität Karlsruhe, Fakultät für Informatik (1995) 10
226. Soldani, J.: Modeling Franklin's Improved Algorithm For Decentralized Extrema Finding In Circular Configurations Of Processors (2014). The CoreASM code for the algorithm can be accessed via https://github.com/szenzaro/WebASM/blob/master/src/main/ide/ExtremaFinding.casm 96, 100, 102
227. Somers, J.: The Coming Software Apocalypse. The Atlantic (2017). URL https://www.theatlantic.com/technology/archive/2017/09/saving-the-world-from-code/540393/. Consulted 01/09/2018 7
228. Stärk, R.F.: Formal specification and verification of the C# thread model. Theoretical Computer Science **343**(3), 482–508 (2005) 10, 141, 293
229. Stärk, R.F., Börger, E.: An ASM Specification of C# Threads and the .NET Memory Model. In: W. Zimmermann, B. Thalheim (eds.) Proc. of 11th International Workshop on Abstract State Machines: Advances in Theory and Practice (ASM 2004), *LNCS*, vol. 3052, pp. 38–60. Springer (2004) 141
230. Stärk, R.F., Nanchen, S.: A Logic for Abstract State Machines. J. Universal Computer Science **7**(11), 980–1005 (2001) 10, 172
231. Stärk, R.F., Schmid, J.: Completeness of a Bytecode Verifier and a Certifying Java-to-JVM compiler. J. of Automated Reasoning **30**(3), 323–361 (203) 10, 293
232. Stärk, R.F., Schmid, J., Börger, E.: Java and the Java Virtual Machine: Definition, Verification, Validation. Springer (2001) VIII, 4, 10, 30, 63, 135, 140, 207, 293, 312, 318
233. Stegmaier, M.: Analyse und Implementierung der erweiterten Semantik eines universellen Kontrollkonstrukts für Abstract State Machines. Master's thesis, Ulm University (2015) 165
234. Stegmüller, M.M.: Formale Verifikation des DLX RISC-Prozessors: Eine Fallstudie basierend auf abstrakten Zustandsmaschinen. Master thesis, University of Ulm (1998) 10, 239
235. Stroetmann, K.: The Constrained Shortest Path Problem: A Case Study In Using ASMs. J. Universal Computer Science **3**(4), 304–319 (1997) 75, 164
236. Tanenbaum, A.S.: Modern Operating Systems: Design and Implementation. Prentice-Hall (1987) 159
237. Teich, J.: Project Buildabong at University of Paderborn. https://www.cs12.tf.fau.de/forschung/projekte/buildabong/ (2001) 239

238. Teich, J., Kutter, P., Weper, R.: Description and Simulation of Microprocessor Instruction Sets Using ASMs. In: Y. Gurevich, P. Kutter, M. Odersky, L. Thiele (eds.) Proc. of International Workshop on Abstract State Machines: Theory and Applications (ASM 2000), *LNCS*, vol. 1912, pp. 266–286. Springer (2000) 239
239. Thalheim, B., Zimmermann, W. (eds.): Proc. of 11th International Workshop on Abstract State Machines: Advances in Theory and Practice (ASM 2004), *LNCS*, vol. 3052. Springer (2004) VI
240. Wainwright, R.T.: Life is Universal. In: Proc. of Winter Simulation Conference (WSC/SIGSIM), vol. 2, pp. 449–459 (1974) 63
241. Wegner, P.: Why Interaction is More Powerful Than Algorithms. Commun. ACM **40**(5), 80–91 (1997) 27
242. Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer (2007) 141
243. Winter, K.: Model Checking Abstract State Machines. Ph.D. thesis, Technical University of Berlin (2001) 293
244. Zenzaro, S.: A CoreASM refinement implementing the ATM ground model. Available at http://modelingbook.informatik.uni-ulm.de (2014) 65, 79, 83
245. Zenzaro, S.: A CoreASM refinement of the P2P ground model. In: S-BPM ONE 2015 Special Session on Comparative Case Studies (2015). Available at http://modelingbook.informatik.uni-ulm.de 205
246. Zenzaro, S.: An ASM model for the Procure To Pay Case Study. In: J. Ehlers, B. Thalheim (eds.) Proc. of 7th International Conference on Subject-Oriented Business Process Management (S-BPM One 2015), S-BPM ONE. ACM (2015) 205
247. Zenzaro, S.: On modularity in Abstract State Machines. Ph.D. thesis, Università di Pisa, Pisa (2016) 135, 137, 165
248. Zenzaro, S., Gervasi, V., Soldani, J.: Web ASM: An Abstract State Machine Execution Environment for the Web. In: Y.A. Ameur, K.D. Schewe (eds.) Proc. of 4th International ABZ Conference ASM, Alloy, B, TLA, VDM, and Z (ABZ 2014), *LNCS*, vol. 8477, pp. 216–221. Springer (2014) 96
249. Zimmerman, W., Gaul, T.: On the Construction of Correct Compiler Back-Ends: An ASM Approach. J. Universal Computer Science **3**(5), 504–567 (1997) 10

Index

E. Börger, A. Raschke, *Modeling Companion for Software Practitioners*,
https://doi.org/10.1007/978-3-662-56641-1