Egon Börger (Pisa)

An ASM Model for an Equipment Rental Business Process

An exercise in Communicating Concurrent ASMs

Università di Pisa, Dipartimento di Informatica, boerger@di.unipi.it

Case Study supplementing Ch.5 of Modeling Companion
http://modelingbook.informatik.uni-ulm.de

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

Goal of the exercise

Exercise the use of concurrent communicating ASMs to construct
a ground model for a tiny commercial business process

- namely the 'Procure-To-Pay' example proposed in the 2013 textbook M. Dumas et al: Fundamentals of Business Process Management

---

Focus of the exercise, inspired by S. Zenzaro's ASM model (2015):

- *translate verbal requirements into an accurate algorithmic form*
  - revealing explicitly process relevant features left implicit by the requs
  - correct incorrect requs, detect & add missing relevant requs

  so that the resulting ground model is checkable, by inspection with domain experts, to 'correctly' and 'completely' express the requs and thus be a reliable basis for a detailed design & implementation
- *illustrate the seamless integratability of control and data* by ASM refinements (instantiations of abstractions for concrete elements)

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

Procure-To-Pay process at BuildIT: Task description

Source: M. Dumas et al: Fundamentals of Business Process Management (Springer 2013). Example 1.1 (p. 2-3)

The text is quoted without change, but we split it into named paragraphs to facilitate relating the requirements to their rigorous model counterpart, during the requirements analysis and for the inspection process which has to verify the model correctness and completeness.

BuildIT is a construction company specialized in public works (roads, bridges, pipelines, tunnels, railroads, etc.). Within BuildIT, it often happens that engineers working at a construction site (called *site engineers*) need a piece of equipment, such as a truck, an excavator, a bulldozer, a water pump, etc. BuildIT owns very little equipment and instead it rents most of its equipment from specialized suppliers.

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

Procure-to-pay: Task description (2)

The existing business process for renting equipment goes as follows.

*ToolRentalReq*uest. When site engineers need to rent a piece of equipment, they fill in a form called 'Equipment Rental Request' and send this request by e-mail to one of the clerks at the companys depot.

*HandleRentalReq*uest. The clerk at the depot receives the request and, after consulting the catalogs of the equipment suppliers, selects the most cost-effective equipment that complies with the request. Next, the clerk checks the availability of the selected equipment with the supplier via phone or e-mail.

HandleAvailability. Sometimes the selected option is not available and the clerk has to select an alternative piece of equipment and check its availability with the corresponding supplier.

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

Procure-to-pay: Task description (3)

HandleApproval. Once the clerk has found a suitable piece of equipment available for rental, the clerk adds the details of the selected equipment to the rental request. Every rental request has to be approved by a works engineer, who also works at the depot.

*ToolReqEval*uation. In some cases, the works engineer rejects the equipment rental request.

HandleRejection. Some rejections lead to the cancellation of the request (no equipment is rented at all). Other rejections are resolved by replacing the selected equipment with another equipment—such as a cheaper piece of equipment or a more appropriate piece of equipment for the job. In the latter case, the clerk needs to perform another availability enquiry.

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

Procure-to-pay: Task description (4)

EngageTool. When a works engineer approves a rental request, the clerk sends a confirmation to the supplier. This confirmation includes a Purchase Order (PO) for renting the equipment. The PO is produced by BuildITs financial information system using information entered by the clerk. The clerk also records the engagement of the equipment in a spreadsheet that is maintained for the purpose of tracking all equipment rentals.

*CancelToolReq*uest. In the meantime, the site engineer may decide that the equipment is no longer needed. In this case, the engineer asks ...

*HandleCancelReq*uest ... the clerk to cancel the request for renting the equipment.

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

Procure-to-pay: Task description (5)

ShipTool. In due time, the supplier delivers the rented equipment to the construction site.

ReceiveTool. The site engineer then inspects the equipment. If everything is in order, the engineer accepts the engagement and the equipment is put into use. In some cases, the equipment is sent back because it does not comply with the requirements of the site engineer. In this case, the site engineer has to start the rental process all over again.

PickUpTool. When the rental period expires, the supplier comes to pick up the equipment.

*ToolRentExtendReq*uest. Sometimes, the site engineer asks for an extension of the rental period by contacting the supplier via e-mail or phone 1–2 days before pick-up.

*AnswerExtensionReq*uest. The supplier may accept or reject this request.

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

Procure-to-pay: Task description (6)

SendInvoice. A few days after the equipment is picked up, the equipments supplier sends an invoice to the clerk by e-mail.

InvoiceCheckReq. At this point, the clerk asks the site engineer to ...

- *ConfirmInvoiceCheck* ... confirm that the equipment was indeed rented for the period indicated in the invoice.

InvoiceCheckCompletion. The clerk also checks if the rental prices indicated in the invoice are in accordance with those in the PO. After these checks, the clerk forwards the invoice to the financial department and ...

- *ProcessInvoice* ... the finance department eventually pays the invoice.

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

The actors of Procure-To-Pay and their communication

- The requirements speak about *five groups of actors* (sets of agents):
  – $SiteEngineer$, $Clerk$, $WorkEngineer$, $Supplier$, $FinanceDept$
- For each $group$ we translate the verbal description of the actions of its actors into a set of ASM rules, constituting the $group\_\text{PROGRAM}$.
  – Each concrete actor executes an instance of its $group\_\text{PROGRAM}$.
- Interaction is by e-mail or phone so that the resulting model PROCURETOPAY will be a system of communicating ASMs.
  – Apparently *reliable communication* ('msg passing') is assumed where no msg gets lost or corrupted and messages arrive in sending order.

Very typically, the analysis of the informal requirements will disclose various missing but process relevant action details and assumptions which become explicit part of the ground model ASM.

- Other additions will appear as possible model refinements.

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

Grouping the required actions into group programs

- *SiteEngineer_Actions* = -- for additional actions see below
  {ToolRentalReq, CancelToolReq,
  ReceiveTool, ToolRentExtendReq$_{1/2}$,
  ConfirmInvoiceCheck, RecordActions}
- *Clerk_Actions* =
  {HandleRentalReq, HandleCancelReq,
  HandleAvailability, HandleApproval,
  HandleRejection, EngageTool, HandleToolRefusal,
  InvoiceCheckReq, InvoiceCheckCompletion}
- *WorkEngineer_Actions* = {ToolReqEval}
- *Supplier_Actions* =
  {AnswerAvailabilityReq, ConfirmToolEngagement,
  ShipTool, AnswerExtensionReq, PickUpTool,
  SendInvoice, PickUpToolRefused}
- *FinanceDept_Actions* = {ProcessInvoice}

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

Sequentiality of actors (single agents)

- It seems reasonable to assume that each actor (a single person) performs in each moment only one action, but different actors may simultaneously make a step. This corresponds exactly to the concept of concurrent ASMs: families of single-agent ASMs which asynchronously execute each one its program, step by step ('sequentially', one step after the other) but possibly together with steps of other agents.

Therefore, for each group $Agent$ the $Agent\_\textsc{Program}$, of which each group member $actor \in Agent$ executes an instance, can be defined as executing in each step one of the $Agent\_Actions$ (read: ASM rules).

$Agent\_\textsc{Program} =$

choose $rule \in Agent\_Actions$ **do** $rule$

In this way we a) *separate scheduling* of single agent actions (which is not considered by the requirements) from modeling the action functionality and b) can *model each action independently* of the others.

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

Analysis of ToolRentalReq requirement

*ToolRentalReq*uest. When site engineers need to rent a piece of equipment, they fill in a form called 'Equipment Rental Request' and send this request by e-mail to one of the clerks....

The *request trigger*—'when site engineers need'—is modeled by the site engineer's choice to execute the $\textsc{ToolRentalReq}$ rule

- namely by accessing the device (computer, mobile phone, etc.) used to submit to the chosen clerk the request for the chosen tool.
- It is required that the request may happen any time. Therefore the rule has no guard.

The *sequentiality* of first filling in the form and then sending the request msg is expressed by the ASM **seq**uential execution operator.

The formatting and classification of msgs is represented by appropriate functions we leave abstract here, like in this case $RentalReqMsg$.

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

Site engineer rule ToolRentalReq

let $request = \mathbf{new}\ (EquipRentalReq)$ -- generate db entry

  let $equip = chosenTool, receiver = chosenClerk$ -- input data

  $tool(request) := equip$

  $clerk(request) := receiver$

  $status(request) := toolRequested$

  $taskId(request) := \mathbf{new}\ (Identifier)$

  CompleteForm$(request)$ -- insert other relevant request data

seq Send$(RentalReqMsg(request), \mathbf{to}\ clerk(request)$

Forms are represented as elements of $EquipRentalReq$. $RentalReqMsg$ extracts and composes the msg content, which we stipulate to

- include $taskId(request)$ and $tool(request)$
- be retrievable from the msg by fcts $taskId(msg), tool(msg)$, etc.

NB. With $taskId$ we use a function $task(taskId(r)) = r$.

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

Missing requirement concerning ToolRentalReq

The *ToolReqEval* requirement states that

> In some cases, the works engineer rejects the equipment rental request. Some rejections lead to the cancellation of the request (no equipment is rented at all).

But the task description says nothing about how this affects the site engineer.

Clearly, the site engineer should at least be informed by a $RentalReqAnswer$ msg (presumably sent by the corresponding clerk) whether his request has been $Granted$ or $Rejected$.

We add this as additional *RecordRentalReqAnsw* requirement and translate it into additional RecordActions rules.

- This example illustrates how to easily insert other request/reply interactions, one by one, into the ASM process model.

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

Two RecordActions RecordRentalReqAnsw$_{+/-}$

if $Received(msg)$ **and** $type(msg) = Req[Accept|Reject]Msg$ **then**
  if thereisno $r \in EquipRentalReq$ **with** $taskId(r) = taskId(msg)$
    then $\textsc{ErrorHandler}(msg)$ -- includes $\textsc{Consume}(msg)$ action
    else let $request = task(msg)$ -- extract the corresponding $request$
      if $GrantedBy(msg, request)$
        then $status(request) := waitingForTool$ -- case +
        else $\textsc{Delete}(request, EquipRentalReq)$ -- case -
      $\textsc{UpdateReqData}(request, msg)$
      $\textsc{Consume}(msg)$
    where $task(msg) =$ -- $\iota r \varphi$ means 'the unique r satisfying φ'
      $\iota r(r \in EquipRentalReq$ **and** $taskId(r) = taskId(msg))$

NB. For documentation purposes, $\textsc{UpdateReqData}(r, msg)$ may include, among other updates, recording a copy of a $Rejected$ request r.

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

The site engineer CancelToolReq rule

*CancelToolReq*uest. In the meantime, the site engineer may decide that the equipment is no longer needed. In this case, the engineer asks the clerk to cancel the request for renting the equipment.

NB. The not furthermore specified decision of a site engineer to cancel a tool request, formulated as a site engineer rule without guard, implies a choice of which request to cancel. The 'In the meantime' condition is expressed by the request status condition $status(req) = toolRequested$. Note that it disallows to cancel an already delivered tool (see below).

CancelToolReq =

choose $request \in EquipRentalReq$ **with**

$status(request) = toolRequested$

Send($CancelReqMsg(request)$, **to** $clerk(request)$)

$status(request) := requestedToBeCanceled$

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

Missing requirement concerning CancelToolReq

- Shouldn't the site engineer receive a $CancelReqAnswMsg$ from the clerk?
- Presumably, whether a tool request can be canceled depends on the stage of its handling by the clerk, whether the tool rental has already been purchased and/or the tool is on its way to the construction site.
  – See the discussion of the clerk rule HandleCancelReq below.

Adding this requirement leads to add two RecordActions a site engineer may perform, say $\text{RecordCancelReqAnsw}_{+/-}$ which can be defined analogously to the RecordRentalReqAnsw rules above.

- A further analysis of those rules would reveal that if the cancel request is refused and the tool is already on its way to the construction site, the tool can still be refused when the site engineer does ReceiveTool (see the rule below).

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

Modeling the ReceiveTool requirement

ReceiveTool. The site engineer then—i.e. once the supplier delivers the rented equipment to the construction site—inspects the equipment. If everything is in order, the engineer accepts the engagement and the equipment is put into use. In some cases, the equipment is sent back because it does not comply with the requirements of the site engineer. In this case, the site engineer has to start the rental process all over again.

By the $taskId$ functions we relate the rule input $deliveredTool$ to the request $r \in EquipRentalReq$ which triggered sending the tool and with which the $deliveredTool$ must comply:

$$taskId(deliveredTool) = taskId(r).$$

It is apparently assumed that the corresponding $ReqAcceptMsg$—a confirmation of the request that triggered $deliveredTool$ to be sent—is $Received$ before the tool arrives so that $deliveredTool$ is $Expected$.

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

The site engineer ReceiveTool rule

if $Expected(deliveredTool)$ -- NB. Input param $deliveredTool$

then let $req = \iota r(ExpectedBy(r, deliveredTool))$ -- request retrieval

  if $IsOkFor(req, deliveredTool)$ -- site engineer input

    then $status(req) := toolWorking$

      UpdateReqData$(req, deliveredTool)$

    else $status(req) := toolRefused$

      ShipBack$(deliveredTool)$ -- placeholder for further spec

      Send$(ToolRefusalMsg(deliveredTool, req),$ **to** $clerk(req))$

else ErrorHandler$(NotRequested(deliveredTool))$

where UpdateReqData(r, t) includes $workingTool(r) := t$

$Expected(t)$ **iff forsome** $r \in EquipRentalReq$ $ExpectedBy(r, t)$

$ExpectedBy(r, t)$ **iff**

  $status(r) = waitingForTool$ **and** $taskId(r) = taskId(t)$

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

Analysing the ToolRentExtendRequest requirement

ToolRentExtendReq. Sometimes, the site engineer asks for an extension of the rental period by contacting the supplier via e-mail or phone 1–2 days before pick-up.

- By the *AnswerExtensionReq*uest, the supplier may accept or reject this request. Apparently it is assumed that each $ExtensionReqMsg$ is answered. Thus the site engineer action seems to consist of two parts:
  - sending an $ExtensionReqMsg$ to the supplier and receiving from there an $ExtensionReqAnsw$ msg. We formulate this by two rules $\textsc{ToolRentExtendReq}_i$ $(i = 1, 2)$.
- No further site engineer action is required when the extension request is not granted. So we disallow in this case further extension requests.
- Presumably a further extension request can be sent only after the preceeding one was granted. We add this as requirement.

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

The site engineer ToolRentExtendReq$_1$ rule

ToolRentExtendReq$_1$ =

choose $task \in EquipRentalReq$ **with** $status(task) = toolWorking$

choose $period \in TimeInterval$ -- $period$ is given as input

if $StillInTimeForExtensionReq(task)$ **then**

let $msg = ExtensionReqMsg(t, taskId(task), period)$

$\text{SEND}(msg, \textbf{to } supplier(task))$

$status(task) := (extensionRequested, period)$

An explicit definition of $StillInTimeForExtensionReq(task)$ depends on the detailed representation of $TimeIntervals$, in particular of $rentalPeriod(task)$, and of the current time (e.g. $today$).

- The *ToolRentExtendReq*uirement example states that the interval from $today$ to $endOf(rentalPeriod(task))$ is at least 2 days.

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

The site engineer ToolRentExtendReq$_2$ rule

ToolRentExtendReq$_2$ =

if $Received(msg)$ **and** $type(msg) = ExtensionReqAnsw$ **then**

if thereisno ... **then** ErrorHandler(msg)

else let $req = task(msg)$ -- retrieve the corresponding task

if $GrantedExtensionBy(msg, req)$ **then**

$rentalPeriod(req) := rentalPeriod(req) + snd(status(req))$

$status(req) := toolWorking$ -- also if extension is refused

Consume(msg) -- may include recording a msg copy

- A detailed definition of $GrantedExtensionBy$ is a matter of refining the format of the messages and requests that are involved.
- A ToolRentExtendReq by a phone call instead of e-mail collapses the two rules into one.

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

The ConfirmInvoiceCheck requirement

A few days after the equipment is picked up, the ... clerk ... asks the site engineer to ...

- *ConfirmInvoiceCheck* ... confirm that the equipment was indeed rented for the period indicated in the invoice.

To enable the site engineer to perform the check, an additional requirement is needed:

- the ReceiveTool action must also record the tool arrival time, say by an update $beginOfRental(r) := today$ as part of the UpdateReqData(r, t) submachine, so that the $endOfRental(r)$ day can be correctly calculated using $rentalPeriod(r)$.
  - Apparently it is assumed that only the registered (possibly extended) $rentalPeriod(r)$ counts, not when the site engineer finishes to use the $workingTool(r)$ or when when the tool is in fact picked up.
    - Probably one of the site engineer's RecordActions should be to RecordToolPickUp.

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

The site engineer ConfirmInvoiceCheck rule

ConfirmInvoiceCheck =

if $Received(msg)$ **and** $type(msg) = InvoiceCheckReq$ **then**

  if thereisno ... **then** ErrorHandler(msg) -- as above

  else let $req = task(msg)$

    if $rentalTime(msg) = [beginOfRental(req), endOfRental(req)]$

      then Send($InvoiceCheckAnsw(yes, msg)$, **to** $clerk(req)$)

      else Send($InvoiceCheckAnsw(no, msg)$, **to** $clerk(req)$)

    Consume(msg)

NB. An alternative would add to RecordActions a rule where the site engineer updates the $endOfRental(rq)$ when using the tool comes to an end, instead of calculating the derived function $endOfRental(rq)$ from $beginOfRental(rq)$ and the $rentalPeriod(rq)$.

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

Analysing the HandleRentalReq requirement

*HandleRentalReq*uest. The clerk at the depot receives the request and, after consulting the catalogs of the equipment suppliers, selects the most cost-effective equipment that complies with the request. Next, the clerk checks the availability of the selected equipment with the supplier via phone or e-mail.

It should be clarified whether the requirements ask for two separate steps—first choosing an equipment/supplier pair from the catalog and then checking the equipment availability with the supplier—or whether it can be considered as one process step. Here we have decided to model the second alternative.

- The requirement involves creating a new depot $task \in RentalTask$ which is related to the requested $tool(msg)$ by a $taskId(msg)$ fct
  - This function extracts from the rental request msg the $taskId$ the site engineer associated with the request and included into the request message.

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

Analysing the HandleRentalReq requirement (Cont'd)

- Presumably other task data are of concern, e.g. the rental period (probably to be indicated by the site engineer and clearly needed by the *ToolRentExtendReq*uest), the rental price, the supplier (where the site engineer can send a tool rental extension request), etc., even if they are not mentioned explicitly in the requirements.
- We use an abstract component $\text{RecordOtherData}(equip, suppl)$ that can be refined when other relevant data are requested to be included as part of a clerk's HandleRentalReq action.
- We stipulate $AvailReqMsg(task)$ to include at least the $taskId(task)$ and the chosen $equip$ment
  - probably among other data like the requested period, the site where the tool is needed, the rental price, etc.

NB. We use the same name for fcts with same role and name but arguments of different type, e.g. $task(msg)$ used by site engineers or by clerks.

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

The clerk rule HandleRentalReq

if $Received(msg)$ **and** $type(msg) = RentalReqMsg$ **then**

let $task$ =**new** $(RentalTask)$ -- clerk creates a new $task$

$taskId(task) := taskId(msg)$ -- relating $task$ to the request

$requestedTool(task) := tool(msg)$

$requestor(task) := sender(msg)$ -- the site engineer

choose $(equip, suppl) \in Catalog$ **with** -- find available tool

$BestComply(equip, suppl, tool(msg))$ -- with price comparison

$assignedTool(task) := equip$

$assignedSupplier(task) := suppl$

$status(task) := askedForAvail$

RecordOtherData$(equip, suppl)$ -- subject to refinement

Consume(msg)

seq Send$(AvailReqMsg(task),$ **to** $suppl)$

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

Analysis of the HandleAvailability requirement

HandleAvailability. Sometimes the selected option is not available and the clerk has to select an alternative piece of equipment and check its availability with the corresponding supplier.

The HANDLEAVAILABILITY action describes iterations of the availability check, performed for $UnAvailabilityAnsw$ers.

- A detailed definition of the $UnAvailabilityAnsw$ predicate, stating that the $AvailReqAnsw$ msg is negative, involves details on the format of messages we abstract from here.
- The *HandleAvailability* requirement made it necessary to record as part of the HANDLERENTALREQ action the information on the tool the site engineer requested: $requestedTool(task) := tool(msg)$.
- For the sake of completeness we include a check that the incoming $AvailReqAnsw$ msg is an answer to a previous request.

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

Open HandleAvailability issues

Various questions remain open and should be answered by the requirements owner. To mention a few:

- Will the (iterations of the) availability check terminate with success?
  - In other words: is it assumed that the clerk will always find an appropriate tool to suggest for the approval by the works engineer?
    - Presumably yes. But then the question comes up whether it is realistic that the clerk never interacts with the site engineer to find an appropriate and possibly alternative available tool, as seems to be assumed in the requirements.
      - Into the ASM model such an additional interaction is easily inserted by an appropriate request/reply message exchange between clerk and site engineer.
- Shouldn't the clerk keep a history of failed availability requests?

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

The clerk rule HandleAvailability

if $Received(msg)$ **and** $type(msg) = AvailReqAnsw$ **then**

  if thereisno $task \in RentalTask$ **with**

    $taskId(task) = taskId(msg)$ **then** HandleError(msg)

  else if $UnAvailabilityAnsw(msg)$ **then** -- only unavailability case

    FindAnotherAvailToolFor$(task(msg))$

    Consume(msg)

where FindAnotherAvailToolFor$(t) =$

  choose $(equip, suppl) \in Catalog$ **with**

    $(equip, suppl) \neq (assignedTool(t), assignedSupplier(t))$

      and $BestComply(equip, suppl, requestedTool(t))$

        $assignedTool(t) := equip \quad assignedSupplier(t) := suppl$

  seq Send$(AvailReqMsg(t),$ **to** $suppl)$

$task(msg) = \iota t(t \in RentalTask$ **and** $taskId(t) = taskId(msg))$

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

Analysis of the HandleApproval requirement

HandleApproval. Once the clerk has found a suitable piece of equipment available for rental, the clerk adds the details of the selected equipment to the rental request. Every rental request has to be approved by a works engineer, who also works at the depot.

- Apparently there is only one $worksEngineer$ at the depot. Otherwise the clerk would have to choose one (and probably to record the choice).
- The $ApprovalReqMsg(task)$ is presumably supposed to contain information on the $requestedTool$, the $assignedTool$, maybe also on the $assignedSupplier$ and the requesting $siteEngineer$.

Once the requirements clarify what is intended, a detailed definition of $ApprovalReqMsg(task)$ and of $\textsc{AddToolDetails}(task(msg))$ can be given.

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

The clerk rule HandleApproval

HandleApproval =

if $Received(msg)$ **and** $type(msg) = AvailReqAnsw$ **then**

if thereisno ... **then** HandleError(msg) -- as above

else if $AvailabilityAnsw(msg)$ **then** -- only availability case

AddToolDetails$(task(msg), msg)$

$status(task) := askedForApproval$

Send$(ApprovalReqMsg(task),$ **to** $worksEngineer)$

Consume(msg)

NB. AddToolDetails presumably includes info on the job the tool is needed for, the rental price and other data needed to evaluate the request, some of them recorded by CompleteForm in the site engineer rule ToolRentalReq.

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

Analysis of the HandleRejection requirement

HandleRejection. Some rejections lead to the cancellation of the request (no equipment is rented at all). Other rejections are resolved by replacing the selected equipment with another equipment—such as a cheaper piece of equipment or a more appropriate piece of equipment for the job. In the latter case, the clerk needs to perform another availability enquiry.

Presumably the $workEngineer$ decides whether the rejection is a definite rejection, leading to cancel the request (and presumably informing the site engineer about this decision), or one that asks the clerk to find another appropriate tool for the requested tool.

Is it realistic that the requirements do not consider:

- any interaction with the site engineer to resolve the rejection of a proposed tool, whether definite or not?
- to inform the supplier about the rejection of the offered tool?

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

The clerk rule HandleRejection

HandleRejection =

if $Received(msg)$ **and** $type(msg) = ApprovalReqAnsw$ **then**

if thereisno ... **then** HandleError(msg) -- as above

else if $RejectionAnsw(msg)$ **then** -- only approval rejection case

if $DefiniteRejectionAnsw(msg)$

then HandleDefiniteReject($task(msg)$)

else FindAnotherAvailToolFor($task(msg)$)

Consume(msg)

where HandleDefiniteReject(t) =

Send($ReqRejectMsg(t)$, **to** $requestor(t)$)

Delete($t, RentalTask$) -- keeping a history record of t?

$status(t) := rejected$

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

An ambiguity in the HandleRejection requirement

- The *HandleRejection* requirement states:

  ... Other rejections are resolved by replacing the selected equipment with another equipment ... the clerk needs to perform another availability enquiry.

  What should happen once the clerk has found an available replacement for the rejected equipment?

  – namely by receiving an $AvailabilityAnsw(msg)$ from some supplier

- The *HandleApproval* requirement states:

  Once the clerk has found a suitable piece of equipment available for rental ... Every rental request has to be approved ...

  We interprete this as requiring that also the chosen replacement tool has to be approved by the works engineer.

  – Otherwise, instead of applying HandleApproval upon receiving an $AvailabilityAnsw(msg)$ for a replacement tool, one would have to add for this case another rule EngageReplacementTool.

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

Analysis of the EngageTool requirement

EngageTool. When a works engineer approves a rental request, the clerk sends a confirmation to the supplier. This confirmation includes a Purchase Order (PO) for renting the equipment. The PO is produced by BuildITs financial information system using information entered by the clerk. The clerk also records the engagement of the equipment in a spreadsheet that is maintained for the purpose of tracking all equipment rentals.

NB. To avoid repetitions, we describe the po-generation as clerk action:

- a mere registration action—without any financial approval check by the financial department—accessing the 'financial information system'.
- This abstracts from a standard msg exchange bw the two actors.

NB. A spreadsheet is a refinement of $RentalTask$.

NB. The required po-relevant data are not specified (see INITIALIZE).

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

The clerk rule EngageTool

if $Received(msg)$ **and** $type(msg) = ApprovalReqAnsw$ **then**
  if thereisno ... **then** HandleError(msg) -- as above
  else if $ApprovingAnsw(msg)$ **then** -- only positive approval case
    let $task = task(msg)$
    $status(task) := approved$
    Send($ReqAcceptMsg(task), requestor(task)$)
    let po =**new** $(PurchaseOrder)$ $po(task) := po$
      Initialize(po) -- updates to insert $task$ data into po
      seq Send($PurchaseOrderMsg(po)$,
                to $assignedSupplier(task)$)
    Consume(msg)

NB. Initialize must be specified to contain all po-relevant $task$ data.

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

The clerk rule HandleCancelReq requirement

*HandleCancelReq*uest. ...(the site engineer asks) the clerk to cancel the request for renting the equipment.

- Nothing is said about how the clerk should handle a task $CancelReqMsg$ which is received (too?) late, e.g. when the tool has already been approved and purchased (and may already be on its way to the site engineer).
  - We choose here to reject such a $CancelReq$uest. Other options are possible, the requirements owner must decide.
- Presumably it is considered as $StillPossibleToCancel(task)$ during the approval phase (before the EngageTool action has been executed).

The requirements must clarify the intended meaning of (and therefore the definition of the predicate) $StillPossibleToCancel(task)$.

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

The clerk rule HandleCancelReq

HandleCancelReq =

if $Received(msg)$ **and** $type(msg) = CancelReqMsg$ **then**

if thereisno ... **then** HandleError(msg) -- as above

else let $task = (task(msg))$

if $StillPossibleToCancel(task)$ **then**

Cancel($task$)

Send($CancelConfirmMsg(task)$, **to** $sender(msg)$)

else Send($TooLateToCancelMsg(task)$, **to** $sender(msg)$)

Consume(msg)

Cancel($task$) should presumably include stopping a still ongoing availability check or approval request processes.

- Exercise: formulate the needed rules for the request/reply interaction between the canceling clerk and suppliers or the $worksEngineer$.

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

Analysis of the InvoiceCheckReq requirement

InvoiceCheckReq. When the equipments supplier sends an invoice to the clerk... the clerk asks the site engineer to confirm that the equipment was indeed rented for the period indicated in the invoice.

Apparently it is assumed that the site engineer will answer.

NB. Since by the *InvoiceCheckCompletion* requirement 'the clerk also checks if the rental prices indicated in the invoice are in accordance with those in the PO', it would be more efficient to perform that check first. But we follow the order indicated in the requirements.

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

The clerk rule InvoiceCheckReq

InvoiceCheckReq =

if $Received(msg)$ **and** $type(msg) = InvoiceMsg$ **then**

if thereisno ... **then** HandleError(msg) -- as above

else let $task = task(msg)$

$invoice(task) := invoiceData(msg)$ -- record invoice msg

$status(task) := invoiceCheck$

Send$(InvoiceCheckReq(task, rentalTime(msg)),$

to $requestor(task))$

Consume(msg)

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

Analysis of the InvoiceCheckCompletion requirement

InvoiceCheckCompletion. The clerk also checks if the rental prices indicated in the invoice are in accordance with those in the PO. After these checks, the clerk forwards the invoice to the financial department ...

- To avoid an irrelevant sequentialization in the specification, we let the clerk check the prices in parallel with looking at the $InvoiceCheckAnsw$ from the site engineer.
- No requirement informs about what should be done to HANDLECORRECTION of an invoice with wrong rental time period or wrong price.
  - Therefore one must either leave these two submachines abstract or agree about additional requirements to refine the submachines.

NB. Further requirements are needed here (and in other cases) if one wants to guarantee that every task is eventually 'completed'.

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

The clerk rule InvoiceCheckCompletion

if $Received(msg)$ **and** $type(msg) = InvoiceCheckAnsw$ **then**
  if thereisno ... HandleError(msg) -- error check as above
  else let $task = task(msg)$
    if $checkResult(msg) = no$ **then**
      HandleCorrection($invoice(task), msg, time$)
      $status(task) := invoiceTimeCorrection$
    else let $res = priceCheck(price(invoice(task)), price(po(task)))$
      if $res = ok$
        then Send($InvoiceMsg(invoice(task))$, **to** $finDept$)
        else HandleCorrection($invoice(task), msg, price$)
          $status(task) := invoicePriceCorrection$
    Consume(msg)

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

The clerk rule to HandleToolRefusal

There are no requirements about the actions involved by a tool refusal.

- We assumed in the ReceiveTool rule that the site engineer informs the clerk about a $ToolRefusalMsg$.
  - Is the site engineer requested to state the reasons (in the msg?) for the refusal?
  - Who should respond to the shipping cost?
- We leave it to the requirements owner to state how to Terminate the rental task: informing the supplier by a $RefusedToolMsg$ (to take back the tool), resolving the shipping cost issue, etc.

HandleToolRefusal =

if $Received(msg)$ **and** $type(msg) = ToolRefusalMsg$ **then**

$status(task(msg)) := toolRefused$

$\text{Terminate}(task(msg), refusal, msg)$

$\text{Consume}(msg)$

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

The *worksEngineer* rule ToolReqEval

*ToolReqEval*uation. In some cases, the works engineer rejects the equipment rental request.

The *HandleRejection* requirements forsee that the *worksEngineer* decides between *approving*, *rejection* or *definiteRejection*, but making the reasons for the decision explicit (or at least documented) is not required. Such a (realistically unspecified?) decision action, the reasons for which remain in the head of the *worksEngineer*, is easily expressed by the ASM **choose** construct.

ToolReqEval =

if $Received(msg)$ **and** $type(msg) = ApprovalReqMsg$ **then**

choose $answ \in \{approving, rejection, definitRejection\}$

Send($ApprovalReqAnsw(answ, task(msg))$, **to** $sender(msg)$)

Consume(msg)

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

The supplier AnswerAvailabilityReq rule

if $Received(msg)$ **and** $type(msg) = AvailReqMsg$ **then**
  if $ThereIsNoToolAvailableFor(tool(msg))$ **then**
    $\text{Send}(AvailReqAnsw(no, msg), \textbf{to}\ sender(msg))$
    else choose $t \in Tool$ **with** $AvailableFor(tool(msg), t)$
      let $task$ =**new** $(ToolTask)$
      $\text{Initialize}(task, t, msg)$ -- record data for offered tool
      $\text{Send}(AvailReqAnsw(yes, msg), \textbf{to}\ sender(msg))$
  $\text{Consume}(msg)$

where

$ThereIsNoToolAvailableFor(t)$ **iff**
  thereisno $t' \in Tool$ **with** $AvailableFor(t, t')$

$AvailableFor(t, t')$ **iff** $status(t') = free$ **and** $IsSpecimenOf(t, t')$

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

Are there any requirements for INITIALIZE$(task, t, msg)$?

INITIALIZE$(task, t, msg)$ should presumably include some of the following updates:

- $status(t) := reserved$ to reserve a specimen of the offered tool
  - probably with inserting values for additional $task$ attributes, which may also be parameters of the $AvailReqAnsw$
    - e.g. construction site address or the validity period of the offer
- $offerRequest(task) := msg$ to record the msg, or at least the relevant data of the request, which triggered the offer
  - What about recording request data for statistical purposes?
- $taskId(task) := taskId(msg)$
- Could it be that if there is no tool available for the request, the supplier wants to buy one or to try to rent one to be able to satisfy the request?

The requirements owner will have to decide upon what should be implemented. Such requirements are then easily integrated by refining INITIALIZE$(task, t, msg)$.

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

Supplier rule ConfirmToolEngagement

This action seems to be implicitly intended by the requirements.
Additional requirements for this action would lead to a rule refinement.

ConfirmToolEngagement =

if $Received(msg)$ **and** $type(msg) = PurchaseOrderMsg$ **then**

if thereisno $task \in ToolTask$ **with**

$taskId(task) = taskId(msg)$ **then** HandleError(msg)

else let $task = task(msg)$

$status(task) := prepareForShipping$

$status(tool(task)) := ordered$ -- commit tool for rental

$clerk(task) := sender(msg)$ -- record who sent the po

PrepareForShipping$(task, msg)$ -- e.g. set $timeToShip$

Send$(OrderConfirmationMsg(msg),$ **to** $sender(msg))$

Consume(msg)

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

The supplier rule ShipTool

ShipTool. In due time, the supplier delivers the rented equipment to the construction site.

Obviously, to 'deliver the equipment' implies to also update the task entry in the database, in particular by the new $status$, presumably to also set a $pickUpTime$ and other relevant data, etc.

Further details can be taken into account by refining this rule.

choose $task \in ToolTask$ **with**
  $status(task) = prepareForShipping$
if $TimeToShip(tool(task))$ **then** -- uses $timeToShip$ attribute
  $status(task) := toolShipped$
  TriggerPhysicalShipping($tool(task)$, **to** $destination(task)$)
  SetPickUpTime&OtherAttributes($task$)

NB. $destination(task)$ describes the address of the construction site where the tool is expected.

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

The supplier rule PickUpToolRefused

The clerk action to HandleToolRefusal presumably involves a corresponding supplier action to ResolveToolRefusal. Additional requirements for this action would lead to a refinement of this rule.

PickUpToolRefused =

if $Received(msg)$ **and** $type(msg) = ToolRefusalMsg$ **then**

if thereisno ... **then** HandleError(msg) -- as above

else let $task = task(msg)$

$status(task) := toolRefused$

$status(tool(task)) := refused$

TriggerPhysicalBringBack$(tool(task))$

ResolveToolRefusal$(task, msg)$

Consume(msg)

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

Analysis of the supplier requirement AnswerExtensionReq

*AnswerExtensionReq*uest. The supplier may accept or reject this request.

- The decision to either accept or reject the request is an input to the process step, therefore represented by a parameterized input variable (read: a monitored function) $decisionToExtend$.
  - The decision may depend not only on the tool and the requested period, but for example also on the time the request arrives, on whether for the requested period extension the status of the tool is still $free$, etc.
    - NB. The latter case would imply that the $status$ function for tools, which is used in the AnswerAvailabilityReq rule, must have a time period as additional parameter.
- An update of the task data involves at least an update of the pick up time by the granted period, may other data like the time of the rental request etc.

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

The supplier rule AnswerExtensionReq

AnswerExtensionReq =

if $Received(msg)$ **and** $type(msg) = ExtensionReqMsg$ **then**

if thereisno ... **then** HandleError(msg) -- as above

else let $task = task(msg)$

let $d = decisionToExtend(tool(task), period(msg), \ldots)$

if $d = yes$ **then**

Send($ExtensionReqAnsw(yes, msg)$, **to** $sender(msg)$)

$status(task) := toolExtensionGranted$

else Send($ExtensionReqAnsw(no, msg)$, **to** $sender(msg)$)

$status(task) := toolExtensionRefused$

UpdateTaskDataBy($task, (d, msg)$)

Consume(msg)

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

The supplier rule PickUpTool

PickUpTool. When the rental period expires, the supplier comes to pick up the equipment.

NB. Presumably further $PickUpAttributes$ are involved in the PhysicalPickUpAction, implying that further database updates may be required as part of the PickUpTool rule.

PickUpTool =

choose $task \in ToolTask$ **with** $status(task) = toolShipped$

if $TimeToPickUp(tool(task))$ **then**

$status(task) := toolToBePickedUp$

TriggerPhysicalPickUpAction$(tool(task), task)$

UpdatePickUpData$(task)$

NB. This rule uses the $pickUpTime$ the ShipTool rule should be (and by us is) assumed to initialize.

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

The SendInvoice requirement

SendInvoice. A few days after the equipment is picked up, the equipments supplier sends an invoice to the clerk by e-mail.

- This implies an additional supplier rule RecordToolReturn to update the task data with the information that the tool came back.
- Such a rule presumably takes also into account that the tool may come back damaged, case in which probably a DamageResolution procedure is started.
  - The requirements owner has to decide whether such cases should be considered.
- Since the requirements do not speak about the invoice data, we specify them abstractly using an *invoice* function. Concrete data can be inserted refining the model by an additional rule which describes how to prepare an invoice for a task.

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

The supplier rule SendInvoice

SendInvoice =

choose $task \in ToolTask$ **with**

$status(task) = toolCameBackOk$

let $inv = invoice(task)$ -- prepare the invoice

Send($InvoiceMsg(inv)$, **to** $clerk(task)$)

$status(task) := invoiceSent$

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

The finance department rule ProcessInvoice

ProcessInvoice ... the finance department eventually pays the invoice.

This requirement is rather 'abstract' and has to be detailed further (which will come up to refine the abstract Pay machine in the model).

It is also probable that the finance department does more than simply pay without further control and without being involved in the permission phase (prior to sending out the purchase order).

ProcessInvoice =

if $Received(msg)$ **and** $type(msg) = InvoiceMsg$ **then**

Pay($invoice(msg)$)

Consume(msg)

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

Analysis of rental request lifetime

The lifetime of a rental request is determined by the sequence of operations performed by corresponding agents to

- request renting a tool,
- procure the tool,
- ship the tool to the construction site,
- operate with the tool at the construction site,
- pay for the rental (invoice operation).

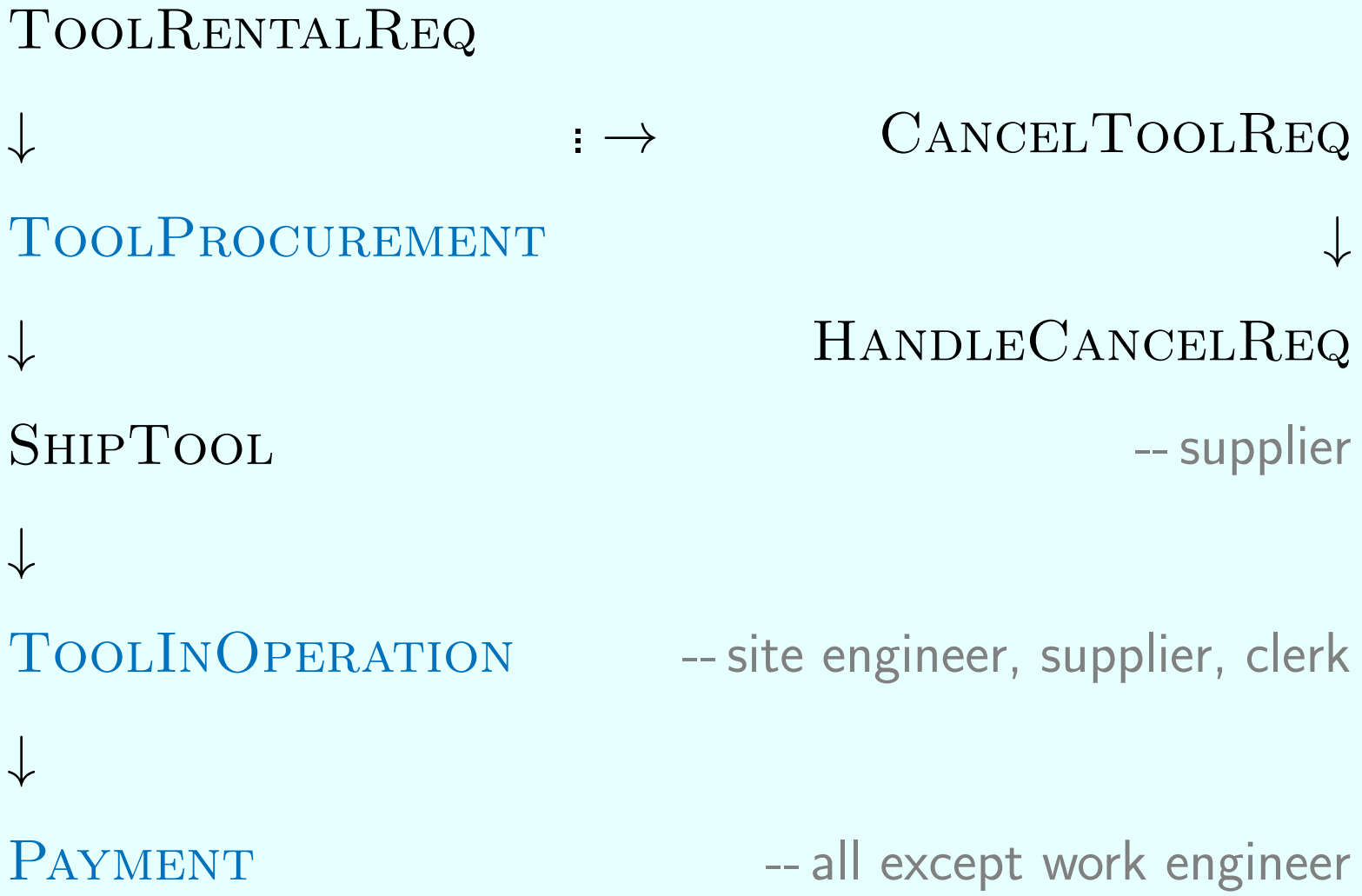
Each of these operations implies a corresponding $status$ of the request.

One can illustrate the possible sequences of rental request lifetime phases by a control flow diagram of the operation calls whose execution determines the phases.

NB. We omit the visualization of the operation guards (the triggers which reflect the data integration into the control flow structure).

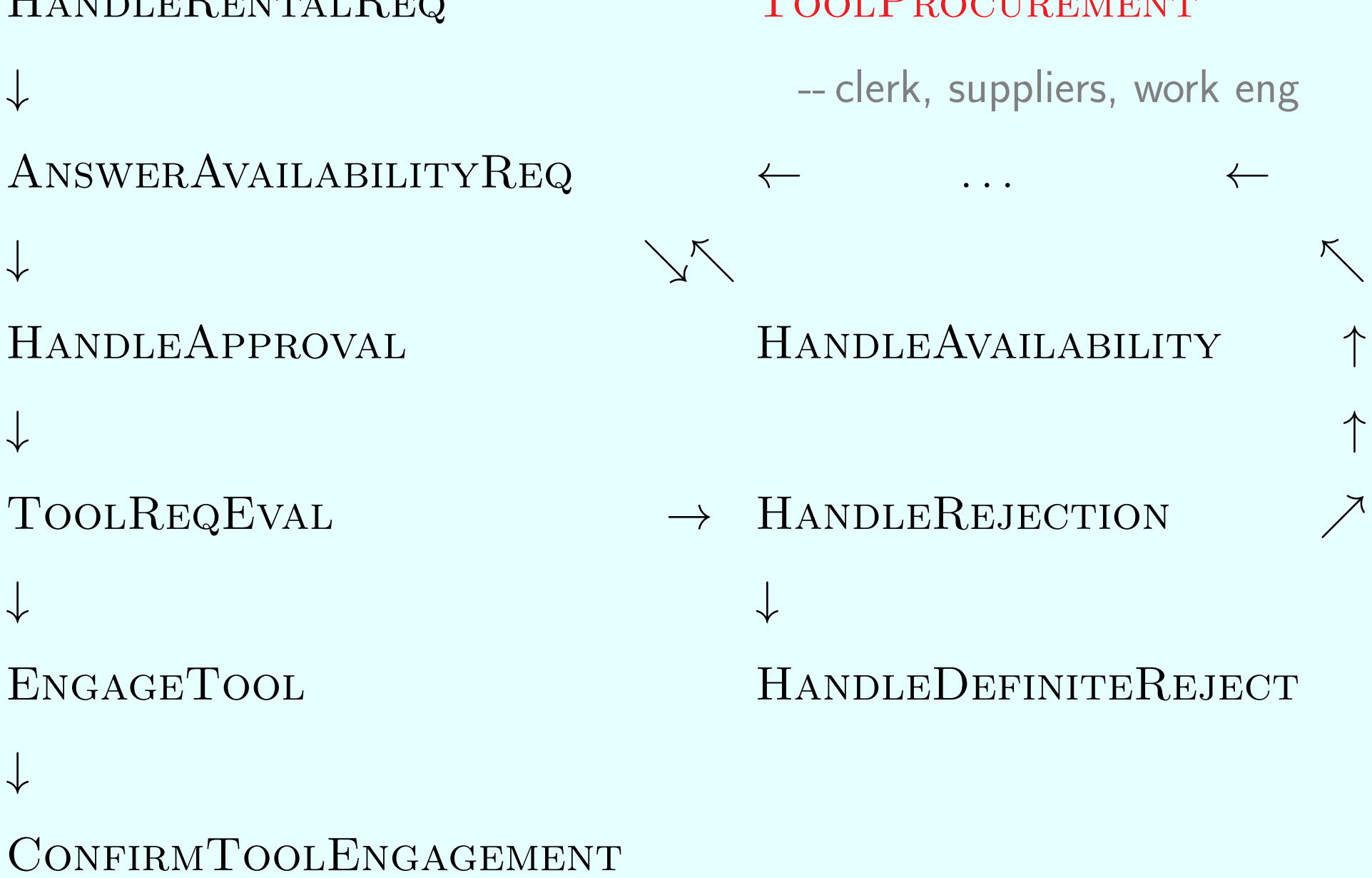
Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

Flow of operations determining rental request lifetime phases



The three macros are defined as follows:

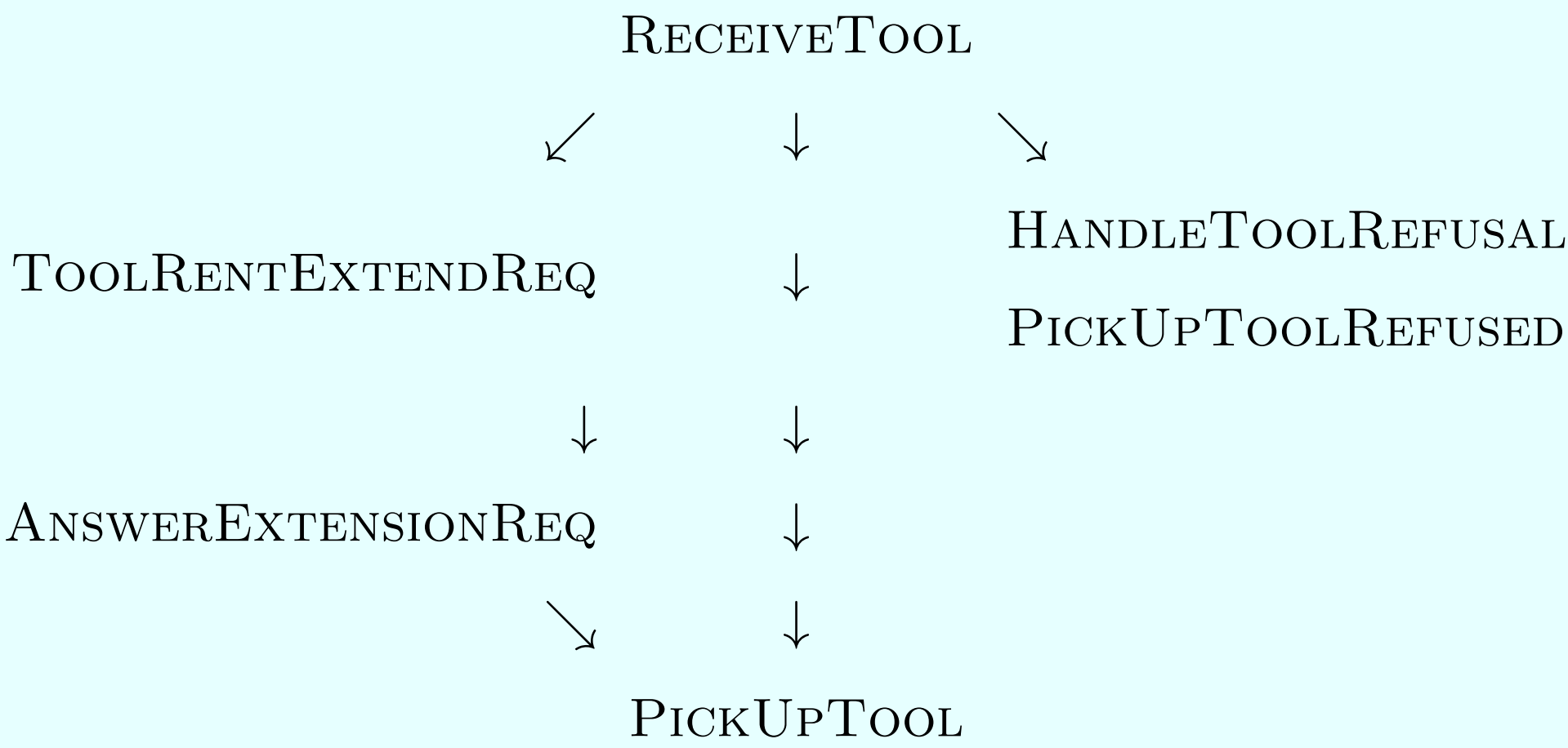
Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0



Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

Sequence of ToolInOperation actions

performed by site engineer, supplier, clerk



Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

Sequence of PAYMENT actions

SENDINVOICE -- supplier action

↓

INVOICECHECKREQ -- clerk action

↓

CONFIRMINVOICECHECK -- site engineer action

↓

INVOICECHECKCOMPLETION → HANDLECORRECTION

↓ -- clerk actions

PROCESSINVOICE -- financial dept action

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

Conclusion: two characteristics of modeling with ASMs

Despite of the rather elementary character of the example—which exhibits mainly only a sequential execution structure combined with the simple request/response pattern—the exercise illustrates two characteristics of modeling with ASMs:

- Abstraction by ASMs supports piecemeal translating verbally formulated requirements into an accurate algorithmic form
  - one by one, componentwise, in the example following the classification of actions of (groups of) actors
- ASM refinement supports
  - data integration into control flow models
  - documenting design ideas which are used to implement abstractions
    - documentation serves explanation, validation and verification

ASM models for some challenging business processes can be found in Ch.5 of the ModelingCompanion book.

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

References

- M. Dumas et al: Fundamentals of Business Process Management (Springer 2013)
  - The book where the case study is proposed.
- S. Zenzaro: An ASM model for the Procure To Pay Case Study. ACM Proc. S-BPM ONE 2015, April 23-24, 2015, Kiel, Germany. http://dx.doi.org/10.1145/2723839.2723865
  - In this work, which inspired the model of this lecture, a similar ASM model and its refinement to a machine-executable CoreASM model are defined.
- E. Börger and A. Raschke: Modeling Companion for Software Practitioners. Springer 2018 http://modelingbook.informatik.uni-ulm.de
  - Communicating ASMs are defined in Ch. 3.

Copyright Egon Börger, released under Creative Commons BY-NC-SA International Licence 4.0

Copyright Notice

It is permitted to (re-) use these slides under the CC-BY-NC-SA licence

https://creativecommons.org/licenses/by-nc-sa/4.0/

i.e. in particular under the condition that

- the original authors are mentioned
- modified slides are made available under the same licence
- the (re-) use is not commercial