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)
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.
The existing business process for renting equipment goes as follows.

**ToolRentalRequest.** 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 company’s depot.

**HandleRentalRequest.** 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.
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.

ToolReqEvaluation. 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.
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.

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

**ToolRentExtendRequest.** 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.

**AnswerExtensionRequest.** The supplier may accept or reject this request.
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.
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_PROGRAM*.
  - Each concrete actor executes an instance of its *group_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.
Grouping the required actions into group programs

- **SiteEngineer**.\(\texttt{Actions} = \) -- for additional actions see below
  \{\texttt{ToolRentalReq}, \texttt{CancelToolReq}, \texttt{ReceiveTool}, \texttt{ToolRentExtendReq}_{1/2}, \texttt{ConfirmInvoiceCheck}, \texttt{RecordActions}\}

- **Clerk**.\(\texttt{Actions} = \)
  \{\texttt{HandleRentalReq}, \texttt{HandleCancelReq}, \texttt{HandleAvailability}, \texttt{HandleApproval}, \texttt{HandleRejection}, \texttt{EngageTool}, \texttt{HandleToolRefusal}, \texttt{InvoiceCheckReq}, \texttt{InvoiceCheckCompletion}\}

- **WorkEngineer**.\(\texttt{Actions} = \) \{\texttt{ToolReqEval}\}

- **Supplier**.\(\texttt{Actions} = \)
  \{\texttt{AnswerAvailabilityReq}, \texttt{ConfirmToolEngagement}, \texttt{ShipTool}, \texttt{AnswerExtensionReq}, \texttt{PickUpTool}, \texttt{SendInvoice}, \texttt{PickUpToolRefused}\}

- **FinanceDept**.\(\texttt{Actions} = \) \{\texttt{ProcessInvoice}\}
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 $\text{Agent}$ the $\text{Agent\_Program}$, of which each group member $\text{actor} \in \text{Agent}$ executes an instance, can be defined as executing in each step one of the $\text{Agent\_Actions}$ (read: ASM rules).

$\text{Agent\_Program} =$

choose $\text{rule} \in \text{Agent\_Actions}$ do $\text{rule}$

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.
Analysis of **ToolRentalReq** requirement

*ToolRentalReq.* 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 **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 *sequential* execution operator.

The formatting and classification of *msgs* is represented by appropriate functions we leave abstract here, like in this case *RentalReqMsg*. 
let request = new (EquipRentalReq) -- generate db entry

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

tool(request) := equip

clerk(request) := receiver

status(request) := toolRequested

taskId(request) := new (Identifier)

COMPLETEFORM(request) -- insert other relevant request data

seq SEND(RentalReqMsg(request), 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.
The \textit{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 \textit{RentalReqAnswer} msg (presumably sent by the corresponding clerk) whether his request has been \textit{Granted} or \textit{Rejected}.

We add this as additional \textit{RecordRentalReqAnsw} requirement and translate it into additional \textit{RecordActions} rules.

- This example illustrates how to easily insert other request/reply interactions, one by one, into the ASM process model.
if $\text{Received}(msg)$ and $\text{type}(msg) = \text{Req}[\text{Accept} | \text{Reject}]\text{Msg}$ then

if there is no $r \in \text{EquipRentalReq}$ with $\text{taskId}(r) = \text{taskId}(msg)$

then $\text{ERRORHANDLER}(msg)$ -- includes $\text{CONSUME}(msg)$ action
else let $request = \text{task}(msg)$ -- extract the corresponding request

if $\text{GrantedBy}(msg, request)$

then $\text{status}(request) := \text{waitingForTool}$ -- case +
else $\text{DELETE}(request, \text{EquipRentalReq})$ -- case -

$\text{UPDATEREQDATA}(request, msg)$

$\text{CONSUME}(msg)$

where $\text{task}(msg) = \iota r. \varphi$ means ‘the unique $r$ satisfying $\varphi$’

$\iota r. (r \in \text{EquipRentalReq} \text{ and } \text{taskId}(r) = \text{taskId}(msg))$

NB. For documentation purposes, $\text{UPDATEREQDATA}(r, msg)$ may include, among other updates, recording a copy of a $\text{Rejected}$ request $r$. 
The site engineer CancelToolReq rule

CancelToolRequest. 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 $\text{status}(\text{req}) = \text{toolRequested}$. Note that it disallows to cancel an already delivered tool (see below).

$$\text{CancelToolReq} =$$

\begin{align*}
\text{choose } & \text{request} \in \text{EquipRentalReq with}
\text{status(request) = toolRequested} \\
\text{SEND}( & \text{CancelReqMsg(request)}, \text{to clerk(request)}) \\
\text{status(request)} & := \text{requestedToBeCanceled}
\end{align*}
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 `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).
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 $r \in \textit{EquipRentalReq}$ which triggered sending the tool and with which the *deliveredTool* must comply:

$$\text{taskId}(\textit{deliveredTool}) = \text{taskId}(r).$$

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*. 
The site engineer **ReceiveTool** rule

if $\text{Expected}(\text{deliveredTool})$  

then let $\text{req} = \text{vr}(\text{ExpectedBy}(\text{r}, \text{deliveredTool}))$  

if $\text{IsOkFor}(\text{req}, \text{deliveredTool})$

then $\text{status}(\text{req}) := \text{toolWorking}$

$\text{UPDATEREQDATA}(\text{req}, \text{deliveredTool})$

else $\text{status}(\text{req}) := \text{toolRefused}$

$\text{SHIPBACK}(\text{deliveredTool})$  

else $\text{ERRORHANDLER}(\text{NotRequested}(\text{deliveredTool}))$

where $\text{UPDATEREQDATA}(\text{r}, \text{t})$ includes $\text{workingTool}(\text{r}) := \text{t}$

$\text{Expected}(\text{t})$ iff forsome $\text{r} \in \text{EquipRentalReq ExpectedBy}(\text{r}, \text{t})$

$\text{ExpectedBy}(\text{r}, \text{t})$ iff

$\text{status}(\text{r}) = \text{waitingForTool}$ and $\text{taskId}(\text{r}) = \text{taskId}(\text{t})$
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 AnswerExtensionRequest, 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 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 preceding one was granted. We add this as requirement.
The site engineer ToolRentExtendReq₁ rule

ToolRentExtendReq₁ =

choose task ∈ EquipRentalReq with status(task) = toolWorking
choose period ∈ TimeInterval       -- period is given as input

if StillInTimeForExtensionReq(task) then
    let msg = ExtensionReqMsg(t, taskId(task), period)
    SEND(msg, to supplier(task))
    status(task) := (extensionRequested, period)

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

The ToolRentExtendRequirement example states that the interval from today to endOf(rentalPeriod(task)) is at least 2 days.
The site engineer ToolRentExtendReq2 rule

**ToolRentExtendReq2** =

if Received\((msg)\) and type\((msg)\) = ExtensionReqAnsw then

if thereisno . . . then  ERRORHANDLER\((msg)\)
else let \(req = \text{task}(msg)\) -- 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.
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 $\text{beginOfRental}(r) := \text{today}$ as part of the $\text{UpdateReqData}(r, t)$ submachine, so that the $\text{endOfRental}(r)$ day can be correctly calculated using $\text{rentalPeriod}(r)$.

  – Apparently it is assumed that only the registered (possibly extended) $\text{rentalPeriod}(r)$ counts, not when the site engineer finishes to use the $\text{workingTool}(r)$ or when when the tool is in fact picked up.

- Probably one of the site engineer’s **RecordActions** should be to **RecordToolPickUp**.
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).
Analysing the **HandleRentalReq** requirement

**HandleRentalReq**. 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 \( \text{task} \in \text{RentalTask} \) which is related to the requested \( \text{tool} (\text{msg}) \) by a \( \text{taskId} (\text{msg}) \) function:
  - This function extracts from the rental request \( \text{msg} \) the taskId the site engineer associated with the request and included into the request message.
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 ToolRentExtendRequest), 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 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 equipment — 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.
The clerk rule \texttt{HandleRentalReq}

\begin{itemize}
\item \textbf{if} \texttt{Received}(\texttt{msg}) \textbf{and} \texttt{type}(\texttt{msg}) = \texttt{RentalReqMsg} \textbf{then}
\item \hspace{1cm} \textbf{let} \texttt{task} = \textbf{new} \ (\texttt{RentalTask})
\item \hspace{1cm} \hspace{1cm} \texttt{taskId}(\texttt{task}) := \texttt{taskId}(\texttt{msg})
\item \hspace{1cm} \hspace{1cm} \texttt{requestedTool}(\texttt{task}) := \texttt{tool}(\texttt{msg})
\item \hspace{1cm} \hspace{1cm} \texttt{requestor}(\texttt{task}) := \texttt{sender}(\texttt{msg})
\item \hspace{1cm} \hspace{1cm} \textbf{choose} \ (\texttt{equip}, \texttt{suppl}) \in \texttt{Catalog} \ \textbf{with}
\item \hspace{1cm} \hspace{1cm} \hspace{1cm} \texttt{BestComply}(\texttt{equip}, \texttt{suppl}, \texttt{tool}(\texttt{msg}))
\item \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \texttt{assignedTool}(\texttt{task}) := \texttt{equip}
\item \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \texttt{assignedSupplier}(\texttt{task}) := \texttt{suppl}
\item \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \texttt{status}(\texttt{task}) := \texttt{askedForAvail}
\item \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \textbf{RecordOtherData}(\texttt{equip}, \texttt{suppl}) \ \textbf{-- subject to refinement}
\item \hspace{1cm} \hspace{1cm} \textbf{Consume}(\texttt{msg})
\item \hspace{1cm} \hspace{1cm} \textbf{seq} \ \textbf{SEND}(\texttt{AvailReqMsg(\texttt{task})}, \texttt{to} \ \texttt{suppl})
\end{itemize}
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 **UnAvailabilityAnswers**.

- A detailed definition of the **UnAvailabilityAns** predicate, stating that the **AvailReqAns** 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: \( \text{requestedTool}(\text{task}) := \text{tool}(\text{msg}) \).

- For the sake of completeness we include a check that the incoming **AvailReqAns** msg is an answer to a previous request.
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?
The clerk rule \texttt{HandleAvailability}

\begin{verbatim}
if Received(msg) and type(msg) = AvailReqAnsw then
  if thereisno task ∈ 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) ∈ Catalog with
    (equip, suppl) ≠ (assignedTool(t), assignedSupplier(t))
    and BestComply(equip, suppl, requestedTool(t))
    assignedTool(t) := equip  assignedSupplier(t) := suppl

seq SEND(AvailReqMsg(t), to suppl)

task(msg) = \nu(t ∈ RentalTask and taskId(t) = taskId(msg))
\end{verbatim}
**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 *AddToolDetails(task(msg))* can be given.
The clerk rule \textbf{HandleApproval}

\textbf{HandleApproval} =

\begin{enumerate}
  \item \textbf{if} \textit{Received}(\textit{msg}) \textbf{and} \textit{type}(\textit{msg}) = \textit{AvailReqAnsw} \textbf{then}
  \item \textbf{if} thereis\textit{no} \ldots \textbf{then} \ \textbf{HANDLEERROR}(\textit{msg}) \quad -- \text{as above}
  \item \textbf{else if} \textit{AvailabilityAnsw}(\textit{msg}) \textbf{then}
    \item \textbf{AddToolDetails}(\textit{task}(\textit{msg}), \textit{msg})
    \item \textit{status}(\textit{task}) := \textit{askedForApproval}
    \item \textbf{Send}(\textit{ApprovalReqMsg}(\textit{task}), \textbf{to} \textit{worksEngineer})
    \item \textbf{CONSUME}(\textit{msg})
\end{enumerate}

\textbf{NB.} \textbf{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 \textbf{CompleteForm} in the site engineer rule \textbf{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?
The clerk rule \textbf{HANDLE REJECTION}

\textbf{HANDLE REJECTION} =

\begin{align*}
\text{if } & \text{Received}(msg) \text{ and type}(msg) = \text{ApprovalReqAnsw} \text{ then} \\
& \text{if there is no \ldots then } \text{HANDLE ERROR}(msg) \quad \text{-- as above} \\
& \text{else if } \text{RejectionAnsw}(msg) \text{ then} \quad \text{-- only approval rejection case} \\
& \quad \text{if } \text{DefiniteRejectionAnsw}(msg) \quad \text{-- only approval rejection case} \\
& \quad \quad \text{then } \text{HANDLE DEFINITE REJECT}(\text{task}(msg)) \\
& \quad \quad \text{else } \text{FIND ANOTHER AVAIL TOOL FOR}(\text{task}(msg)) \\
& \text{CONSUME}(msg) \\
\end{align*}

\textbf{where} \text{HANDLE DEFINITE REJECT}(t) =

\begin{align*}
& \text{SEND}(\text{ReqRejectMsg}(t), \text{to requestor}(t)) \\
& \text{DELETE}(t, \text{RentalTask}) \quad \text{-- keeping a history record of } t? \\
& \text{status}(t) := \text{rejected}
\end{align*}
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**.
**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).
The clerk rule **EngageTool**

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

if there is no \ldots then \textbf{HANDLEERROR}(msg) \quad -- \text{as above}

else if $ApprovingAnsw(msg)$ then \quad -- only positive approval case

\begin{verbatim}
let task = task(msg)
status(task) := approved
SEND(ReqAcceptMsg(task), requestor(task))
end if

let po = new (PurchaseOrder) po(task) := po
\end{verbatim}

\textbf{INITIALIZE}(po) \quad -- \text{updates to insert task data into po}

\begin{verbatim}
seq SEND(PurchaseOrderMsg(po),
        to assignedSupplier(task))
\end{verbatim}

\textbf{CONSUME}(msg)

NB. \textbf{INITIALIZE} must be specified to contain all po-relevant task data.
HandleCancelReq. ...(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 CancelRequest. 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).
The clerk rule \texttt{HandleCancelReq}

\texttt{HandleCancelReq} =

\begin{verbatim}
  if \texttt{Received}(msg) \texttt{and type}(msg) = \texttt{CancelReqMsg} then
    if thereisno ... then \texttt{HandleError}(msg)
  else let \texttt{task} = (\texttt{task}(msg))
    if \texttt{StillPossibleToCancel}(\texttt{task}) then
      \texttt{CANCEL}(\texttt{task})
      \texttt{SEND}(\texttt{CancelConfirmMsg}(\texttt{task}), \texttt{to} \texttt{sender}(msg))
    else \texttt{SEND}(\texttt{TooLateToCancelMsg}(\texttt{task}), \texttt{to} \texttt{sender}(msg))
    \texttt{CONSUME}(msg)
\end{verbatim}

\texttt{CANCEL}(\texttt{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 \texttt{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.
The clerk rule \texttt{InvoiceCheckReq}:

\texttt{InvoiceCheckReq} =

\begin{align*}
\text{if } & \text{Received}(msg) \text{ and } \text{type}(msg) = \text{InvoiceMsg} \text{ then} \\
\text{if } & \text{thereisno ... then } \text{HANDLEERROR}(msg) \\
\text{else let } & \text{task} = \text{task}(msg) \\
\phantom{\text{else let }} & \text{invoice}(\text{task}) := \text{invoiceData}(msg) \\
\phantom{\text{else let }} & \text{status}(\text{task}) := \text{invoiceCheck} \\
\phantom{\text{else let }} & \text{SEND(InvoiceCheckReq(task, rentalTime(msg)),} \\
\phantom{\text{else let }} & \phantom{\text{SEND(InvoiceCheckReq(task, rentalTime(msg)),} \\
\phantom{\text{else let }} & \phantom{\text{SEND(InvoiceCheckReq(task, rentalTime(msg)),} \\
\phantom{\text{else let }} & \phantom{\text{SEND(InvoiceCheckReq(task, rentalTime(msg)),} \\
\phantom{\text{else let }} & \phantom{\text{SEND(InvoiceCheckReq(task, rentalTime(msg)),} \\
\phantom{\text{else let }} & \phantom{\text{SEND(InvoiceCheckReq(task, rentalTime(msg)),} \\
\phantom{\text{else let }} & \text{to requestor(task))} \\
\text{CONSUME}(msg) \end{align*}

-- as above

-- record invoice 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 42
The clerk rule **InvoiceCheckCompletion**

```
if Received(msg) and type(msg) = InvoiceCheckAnsw then
    if there is no ... 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)
```
The clerk rule to **HANDLE TOOL REFUSAL**

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

- We assumed in the **RECEIVE TOOL** 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.

**HANDLE TOOL REFUSAL** =

\[
\text{if } \text{Received}(\text{msg}) \ \text{and} \ \text{type}(\text{msg}) = \text{ToolRefusalMsg} \ \text{then}
\]

\[
\text{status}(\text{task}(\text{msg})) := \text{toolRefused}
\]

\[
\text{TERMINATE}(\text{task}(\text{msg}), \text{refusal}, \text{msg})
\]

\[
\text{CONSUME}(\text{msg})
\]
The *worksEngineer rule* $\text{ToolReqEval}$

ToolReqEvalulation. 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.

$\text{ToolReqEval} =$

if $\text{Received}(msg)$ and $\text{type}(msg) = \text{ApprovalReqMsg}$ then

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

Send($\text{ApprovalReqAnsw}(\text{answ}, \text{task}(msg))$, to $\text{sender}(msg)$)

Consume($msg$)
The supplier **AnswerAvailabilityReq** rule

if Received(msg) and type(msg) = AvailReqMsg then
  if ThereIsNoToolAvailableFor(tool(msg)) then
    Send(AvailReqAnsw(no, msg), to sender(msg))
  else choose \( t \in Tool \) with AvailableFor(tool(msg), t)

    let task = \textbf{new} (ToolTask)

    \textbf{Initialize}(task, t, msg) \quad -- \text{record data for offered tool}

    Send(AvailReqAnsw(yes, msg), to sender(msg))

  \textbf{Consume}(msg)

where

**ThereIsNoToolAvailableFor** \( (t) \) iff

\( \text{thereisno} \ t' \in Tool \) with AvailableFor\( (t, t') \)

AvailableFor\( (t, t') \) iff \( \text{status}(t') = \text{free} \) and IsSpecimenOf\( (t, t') \)
Are there any requirements for \texttt{INITIALIZE(task, t, msg)}?

\texttt{INITIALIZE(task, t, msg)} should presumably include some of the following updates:

- \texttt{status(t) := reserved} to reserve a specimen of the offered tool
  - probably with inserting values for additional \texttt{task} attributes, which may also be parameters of the \texttt{AvailReqAnsw}
    - e.g. construction site address or the validity period of the offer
- \texttt{offerRequest(task) := msg} to record the \texttt{msg}, or at least the relevant data of the request, which triggered the offer
  - What about recording request data for statistical purposes?
- \texttt{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 \texttt{INITIALIZE(task, t, msg)}.
Supplier rule $\text{ConfirmToolEngagement}$

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

$\text{ConfirmToolEngagement} =$

\[
\begin{align*}
\text{if } &\text{Received}(msg) \text{ and type}(msg) = \text{PurchaseOrderMsg} \text{ then} \\
\text{if } &\text{there is no } task \in \text{ToolTask with} \\
&\text{taskId}(task) = \text{taskId}(msg) \text{ then } \text{HandleError}(msg) \\
\text{else let } &task = \text{task}(msg) \\
&\text{status}(task) := \text{prepareForShipping} \\
&\text{status}(\text{tool}(task)) := \text{ordered} \quad \text{-- commit tool for rental} \\
&\text{clerk}(task) := \text{sender}(msg) \quad \text{-- record who sent the po} \\
&\text{PrepareForShipping}(task, msg) \quad \text{-- e.g. set timeToShip} \\
&\text{Send}(\text{OrderConfirmationMsg}(msg), \text{to sender}(msg)) \\
&\text{Consume}(msg)
\end{align*}
\]
The supplier rule \textbf{ShipTool}

\textit{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 \textit{status}, presumably to also set a \textit{pickUpTime} and other relevant data, etc.

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

\texttt{choose task ∈ ToolTask with} \quad \texttt{status(task) = prepareForShipping}

\texttt{if TimeToShip(tool(task)) then} \quad \texttt{-- uses timeToShip attribute}

\texttt{status(task) := toolShipped}

\texttt{TRIGGER_PHYSICAL_SHIPPING(tool(task), to destination(task))}

\texttt{SET_PICK_UP_TIME\&OTHER_ATTRIBUTES(task)}

NB. \textit{destination(task)} describes the address of the construction site where the tool is expected.
The clerk action to **HANDLE_TOOL_REFUSAL** presumably involves a corresponding supplier action to **RESOLVE_TOOL_REFUSAL**. 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  HANDLE_ERROR(msg)  -- as above
else let task = task(msg)
    status(task) := toolRefused
    status(tool(task)) := refused
    TRIGGER_PHYSICAL_BRING_BACK(tool(task))
    RESOLVE_TOOL_REFUSAL(task, msg)
    CONSUME(msg)
```
AnswerExtensionRequest. 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.
The supplier rule \texttt{AnswerExtensionReq}

\texttt{AnswerExtensionReq} =

\begin{verbatim}
if Received(msg) and type(msg) = ExtensionReqMsg then
  if thereisno \ldots then \texttt{HANDLEERROR}(msg) \quad -- as above
else let task = task(msg)
  let d = decisionToExtend(tool(task), period(msg), \ldots)
  if d = yes then
    \texttt{SEND}(ExtensionReqAnsw(yes, msg), to \texttt{sender}(msg))
    \texttt{status}(task) := toolExtensionGranted
  else \texttt{SEND}(ExtensionReqAnsw(no, msg), to \texttt{sender}(msg))
    \texttt{status}(task) := toolExtensionRefused
  \texttt{UPDATE}TaskDataBy(task, (d, msg))
\texttt{CONSUME}(msg)
\end{verbatim}
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 ∈ 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.
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.
The supplier rule $\text{SendInvoice}$

$\text{SendInvoice} =$

\begin{align*}
\text{choose } & \text{ task } \in \text{ ToolTask with} \\
& \text{status}(\text{task}) = \text{toolCameBackOk} \\
\text{let } & \text{ inv } = \text{ invoice}(\text{task}) \\
& \text{ Send(InvoiceMsg(inv), to clerk(\text{task}))} \\
& \text{ status(\text{task}) } := \text{ invoiceSent}
\end{align*}
The finance department rule \texttt{ProcessInvoice}

\textit{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 \texttt{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).

\texttt{ProcessInvoice} =

\texttt{if Received}(msg) \texttt{and type}(msg) = \textit{InvoiceMsg} \texttt{then}
\texttt{PAY}(\textit{invoice}(msg))
\texttt{CONSUME}(msg)
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).
Flow of operations determining rental request lifetime phases

\[ \text{ToolRentalReq} \]

\[ \downarrow \quad : \quad \rightarrow \quad \text{CancelToolReq} \]

\[ \text{ToolProcurement} \quad \downarrow \]

\[ \downarrow \quad \text{HandleCancelReq} \quad \downarrow \]

\[ \text{ShipTool} \quad \text{-- supplier} \]

\[ \downarrow \]

\[ \text{ToolInOperation} \quad \text{-- site engineer, supplier, clerk} \]

\[ \downarrow \]

\[ \text{Payment} \quad \text{-- all except work engineer} \]

The three macros are defined as follows:
HandleRentalReq

AnswerAvailabilityReq

HandleApproval

ToolReqEval

EngageTool

ConfirmToolEngagement

ToolProcurement

-- clerk, suppliers, work eng

← ...

HandleAvailability

HandleApproval

HandleRejection

HandleDefiniteReject
Sequence of **ToolInOperation** actions

performed by site engineer, supplier, clerk

```
ReceiveTool

↓    ↓    ↓

ToolRentExtendReq    ↓

↓    ↓

AnswerExtensionReq    ↓

↓    ↓

PickUpTool
```

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

↓

ProcessInvoice → financial dept action
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.
– The book where the case study is proposed.
– 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.
– Communicating ASMs are defined in Ch. 3.
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