**Egon Börger (Pisa)**

# Closing the Gap

## between Business Process Models and their Implementation

## Towards Certified BPMs

Joint work with Albert Fleischmann (Metasonic)

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

# The problem

Frequently experienced mismatch between

- users' understanding of a Business Process (BP)
- behavior of machines which execute the BP

It is not avoided by BPMs designed with standardized notations

- e.g. OMG BPMN 2.0 and 'standard' compilation to OASIS BPEL
  - due to insufficient precision, lack of completeness, conceptual mismatch, etc. (see critical evaluation in J.SSM Sept. 2011)

This is an instance of a well-known general problem to bridge the gap between the two ends of system development:

- human understanding and formulation of real-world problems
- deployment of their solutions by code-executing machines on changing platforms

# The gap: how to match requirements and code?

- Requirement documents are *descriptions of real-world problems and activities*, typically written by domain experts *for system design experts* (usually not knowledgeable in the application domain), formulated in natural language, interspersed with diagrams, tables, formulae, etc. Frequently such descriptions suffer from lack of precision, ambiguity, incompleteness, inconsistency.
- Compilable programs are *software representations* of computer-based systems, written *for mechanical elaboration* by machines (symbol manipulation) and therefore coming with every needed implementation detail (technical precision, completeness, consistency).

- How can (informal) requirements and (formal) code, the latter written to satisfy the former, be linked in a way to certifiably guarantee that the code does what the requirements describe and not something else?
- How can the link between requirements and code be reliably preserved during maintenance (*requirements change*)?

# What is needed to 'bridge the gap'

- a precise general language with a validation framework
  - practicing domain experts & system designers can use in daily work

  to formulate, justify and document prior to coding
  *accurate models* of real-world problems
  (ground model problem)
- a rigorous general design and verification method
  - practicing software system managers and programmers can incorporate into their development environment
    - including system maintenance ('design for change')

  to successively detail (*stepwise implement*) in a controllably correct manner model abstractions down to executable code
  (verifiable-implementation problem)

# What are ground models?

Accurate blueprints of the to-be-implemented piece of real world —called 'golden models' in the semiconductor industry—which

- define 'the conceptual construct/the essence' of the software system (Brooks) prior to coding, *abstractly and rigorously*
  - at an application-problem-determined level of detailing (*minimality*)
  - formulated in application domain terms (*precision*, informal accuracy)
  - authoritatively for the further development activities: design contract/process/evaluation and maintenance (*completeness*)
- ground the design in reality by justifying the definition as
  - *correct*: model elements reliably convey original intentions
  - *complete*: every semantically relevant feature is present (env,arch, domain knowledge), no gap in understanding of 'what to build'
  - *consistent*: conflicting objectives in requirements resolved

# Ground model justification must solve three problems

- Communication (language) problem: mediate between
  - sw designers, domain experts and customers for common understanding prior to coding of 'precisely what to build'
  - problem domain and world of models, requiring
    - capability to calibrate degree of model precision to the problem
    - general data and operation framework and general interface concept (to represent system environments)
- Verification method problem: no infinite regress
  - no math. transition from informal to precise descriptions, BUT
  - inspection can provide evidence of direct correspondence bw ground model and reality the model has to capture (completeness, correctness, empirical interpretation of extra-logical terms)
  - domain-specific reasoning can check consistency issues
- Validation problem: need for repeatable experiments to validate (falsify) model behaviour (runtime verification and analysis, testing)

# Appropriateness of ASMs for building ground models

- flexible expressability of states and state evolution which
  - is arguably most general (ASM Thesis)
  - uses basic language elements, in particular rules of form

    $$\textbf{if } Condition \textbf{ then } Action$$

    - $Action$ assigns vals (of whatever type) to parameterized mem locs

      $$locName(exp_1, \ldots, exp_n) := exp$$

    - $Condition$ is any state expression

  with generally understood intuitive but mathematically precise behavioral semantics
- executability of such rules (conceptually and tool supported) permits experimental validation (simulation, testing, model checking)
- mathematical definition of semantics yields verifiability of properties by proofs (proof sketch, mathematical or machine checked proof)

# Examples of ASMs serving as ground models

- variety of ASM ground models for industrial standards and systems in railway control, telecommunication, programming languages, protocols, business systems, etc. (see AsmBook)
- ground model ASM for Metasonic's Subject-Oriented Business Process Modeling (S-BPM) tool
  - correctly interpretes the BPMs designed by users
  - mediates bw users's application-domain-centric view and implementers's code view
    - of BP defined by the model (using the graphical editor) and executed by the code running machine

  in fact used inhouse for maintenance purposes

# Consequence: one can tailor ASMs specifically for BPM

We define an appropriate class of ASMs based upon which the BP expert can express a BP design using directly BP-knowledge-based (graphically represented) terms/notations which are supported in two directions by:

- underlying ASM constructs expressing their intuitive understanding
  - correctly: for the BP expert, controllably by inspection and validation
  - precisely: for the sw expert as spec of the implementation
- an implementation of the ASM models
  - the correctness of which is (in principle) provable, given the mathematical character of ASM models and their behavior preserving ASM refinements to executable code (see below)

Consequence: the mathematically precise ASM definition of the behavioral semantics of the graphical notations

- can be hidden without loss of reliability to the BP developer who works with the underlying intuitive understanding of the graphical constructs

# Hierarchical BP view in S-BPM

Three stepwise refinable levels of detail reflect the structure of:

- communication links through which subjects (read: behavior-executing agents) interact with each other by exchanging messages,
- behavior of single subjects, i.e. the sequence of individual internal function or communication actions performed by a single subject,
- data, i.e. information about business objects the subjects manipulate locally (by internal functions) or transmit (via message exchange).

NB. Granularity of S-BPMs depends on decision about which subjects explicitly appear as actors of the to-be-defined BP

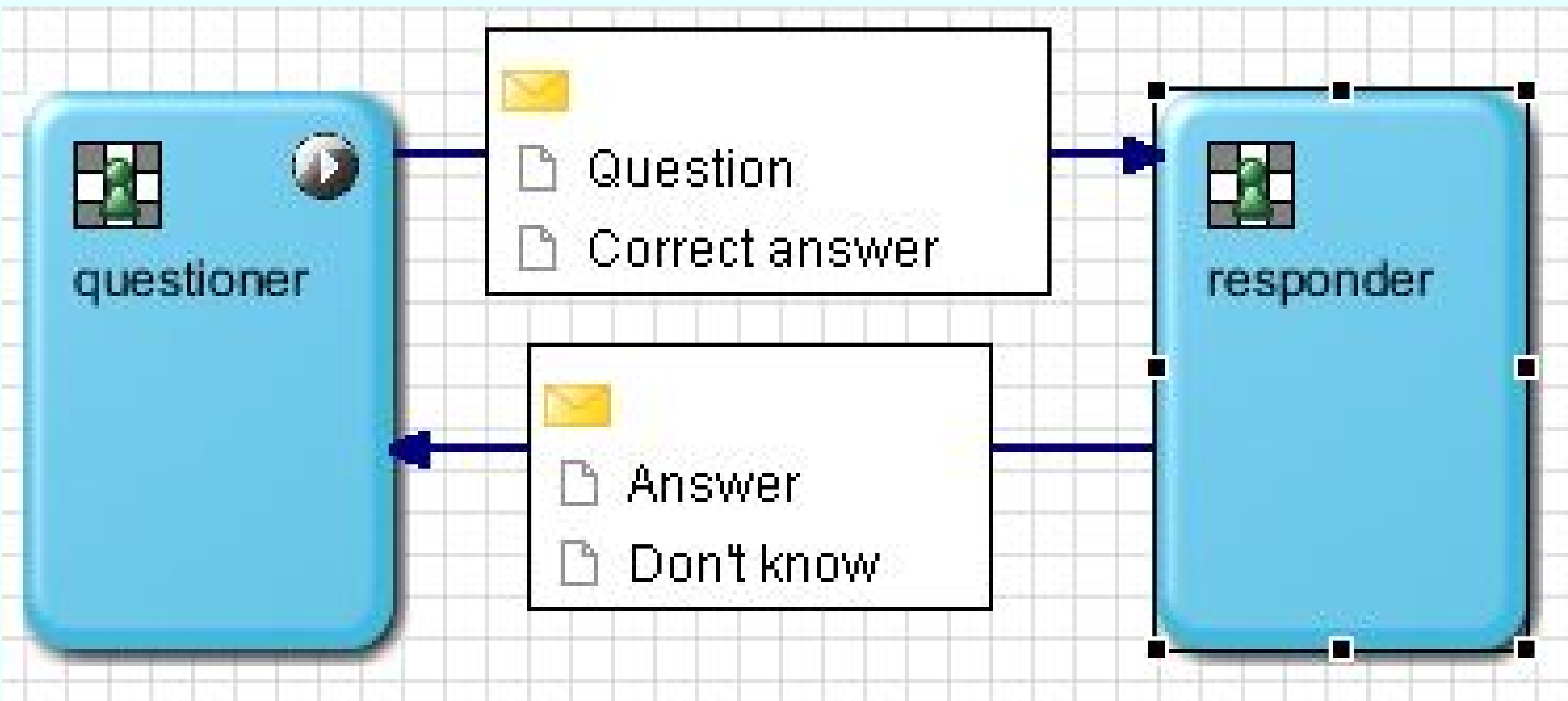
- reflecting particular business needs of a process

# Subject interaction diagrams

Directed graph defining communication structure (signature):

- nodes represent (read: are labeled by) subjects
- directed arcs represent type of messages subjects send resp. receive

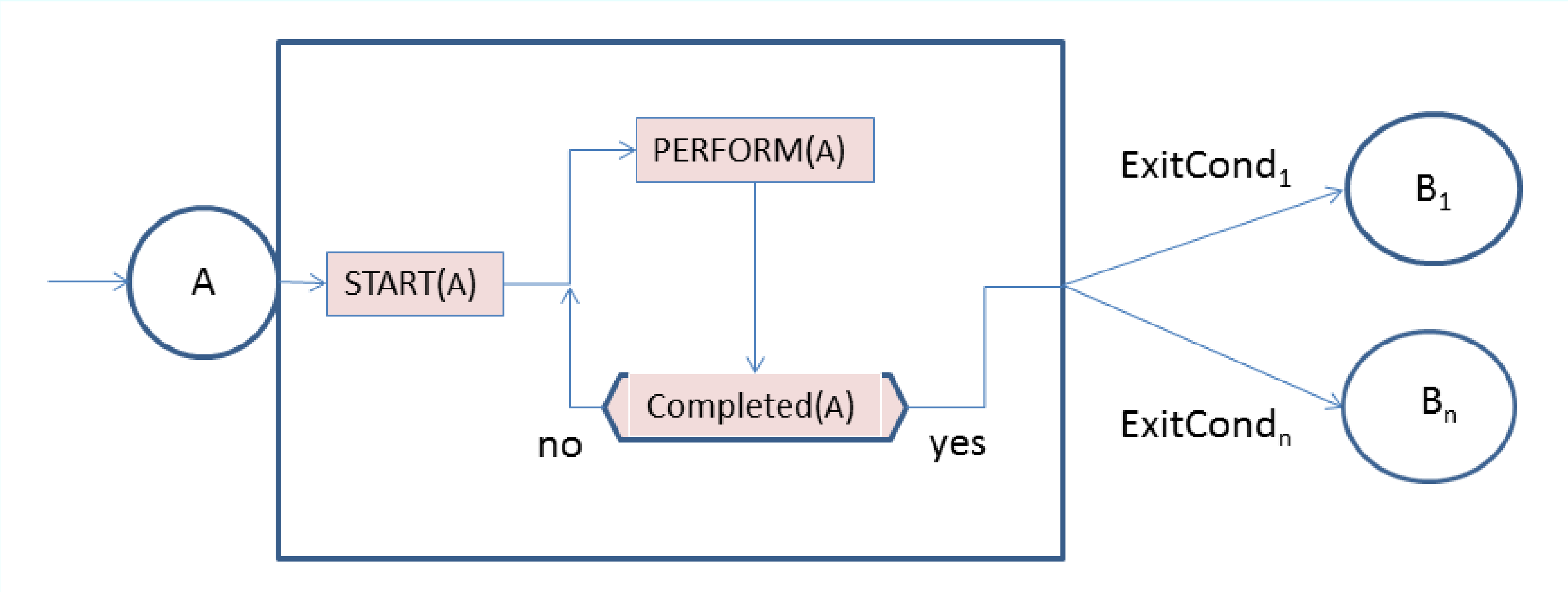
SIDs hide internal actions the subjects can perform and msg content



# Subject Behavior Diagram (SBD)

ASMs with FSM control structure displayed by traditional flowchart of:

- nodes representing
  - internal function $(\textsc{Perform}(A))$ control states
  - *send* control states
  - *receive* control states
- arcs representing *ExitCond*itioned control state transitions



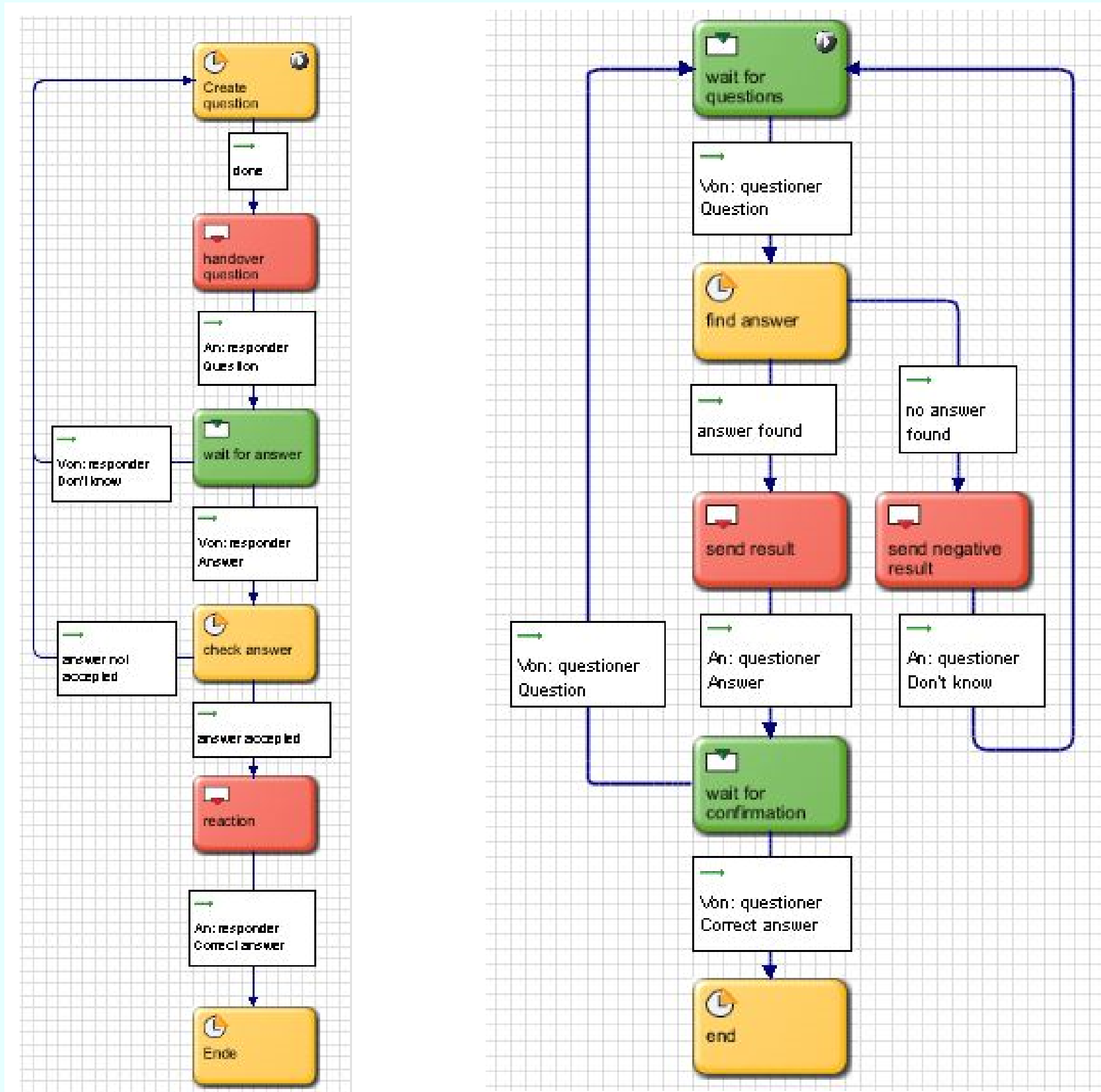
# S-BPM interpreter ASM $\textsc{Behavior}_{BP}$ for a BP

- ASM $\textsc{Behavior}(subj, node)$ describes what a *subj*ect does at *node*
- sequential ASM $\textsc{Behavior}_{subj}(D)$—set of $\textsc{Behavior}(subj, node)$ for all *node*s of $D$—describes what a *subj*ect does when stepping through the SBD from initial to end state
- concurrent ASM $\textsc{Behavior}_{BP}$—set of $\textsc{Behavior}_{subj}(D)$ for all relevant *subj*ects and $D$—describes the behavior of the entire BP

Intuitive understanding of $\textsc{Behavior}(subj, node)$ is accurate relative to understanding of what it means to $\textsc{Start}$ and to $\textsc{Perform}$ the associated service until it is *Completed*—three notions which are defined:

- for internal actions by domain-specific meaning known to BP expert
- for communication actions by specific ASM refinements below for:
  - Sending (synchronous or asynchronous)
  - Receiving (synchronous or asynchronous)



Create question
done
handover question
An: responder Question
wait for answer
Von: responder Don't know
Von: responder Answer
check answer
answer not accepted
answer accepted
reaction
An: responder Correct answer
Ende
wait for questions
Von: questioner Question
find answer
answer found
no answer found
send result
send negative result
Von: questioner Question
An: questioner Answer
An: questioner Don't know
wait for confirmation
Von: questioner Correct answer
end

# Structure of Sending: three-step-refinement definition

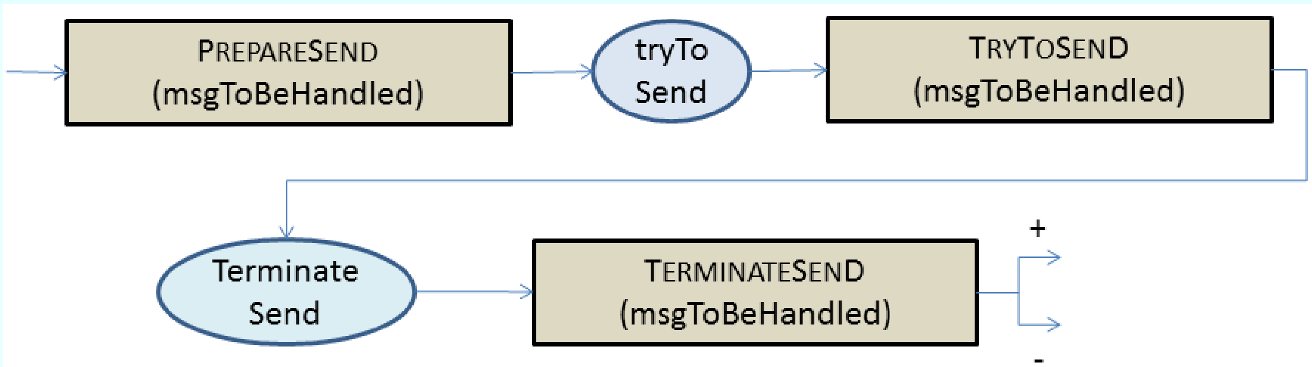
- *single send* action, i.e. sending one message
- *multiple send* action by which a given multitude of messages can be sent as a bundle
- *alternative send* action allowing to repeatedly select among a set of alternatives

NB. Conservative (purely incremental) ASM refinement strongly supports modular design and verification techniques.

S-BPM communication via $inputPool(subject)$ where sender may *deliver* msgs and from where receiver may be ready to *receive* (read: locally store) them

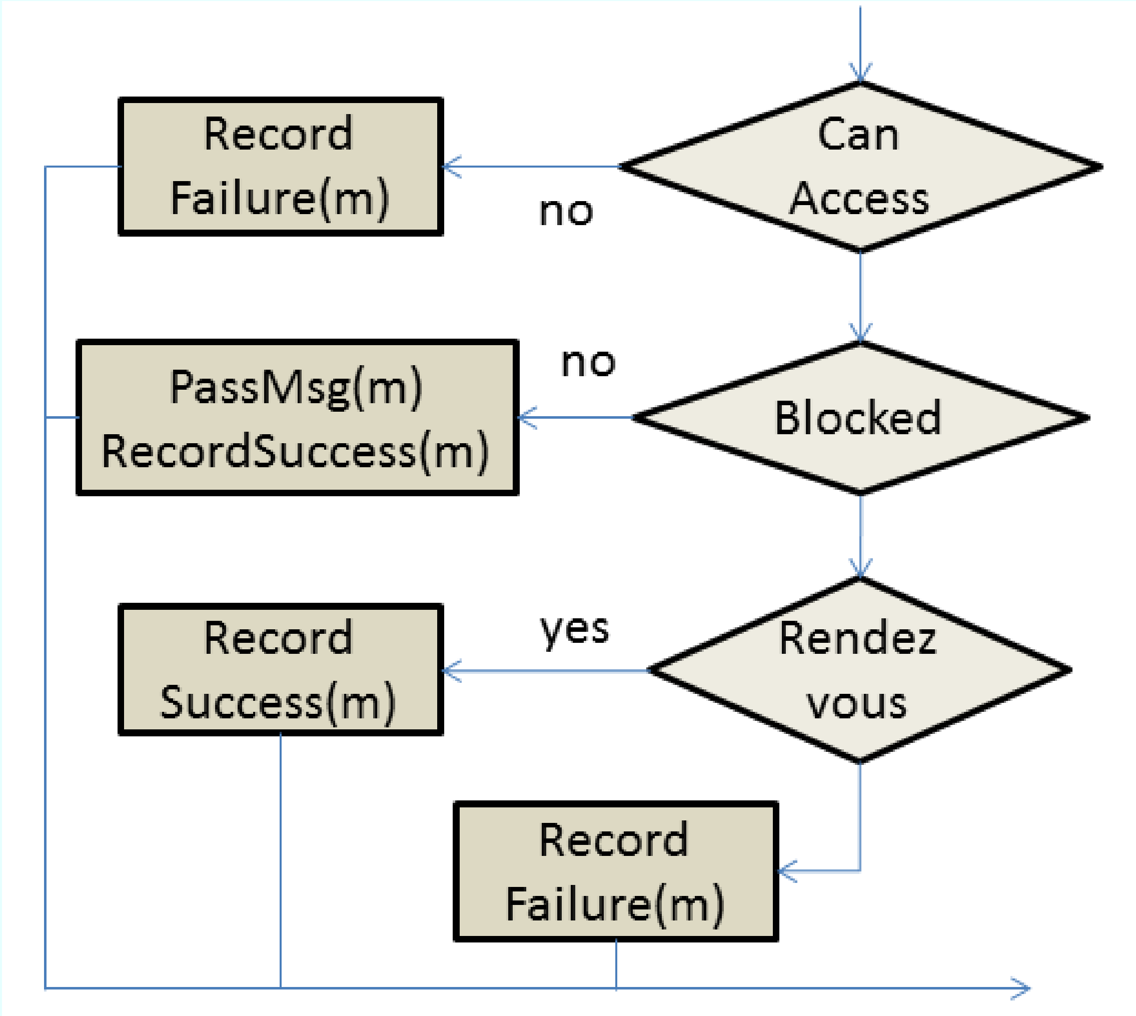
- configurable when input pool is accessible or blocked (for a message of a specific type and/or from a specific sender)
  - to uniformly handle synchronous/asynchronous communication

# The SINGLESEND machine



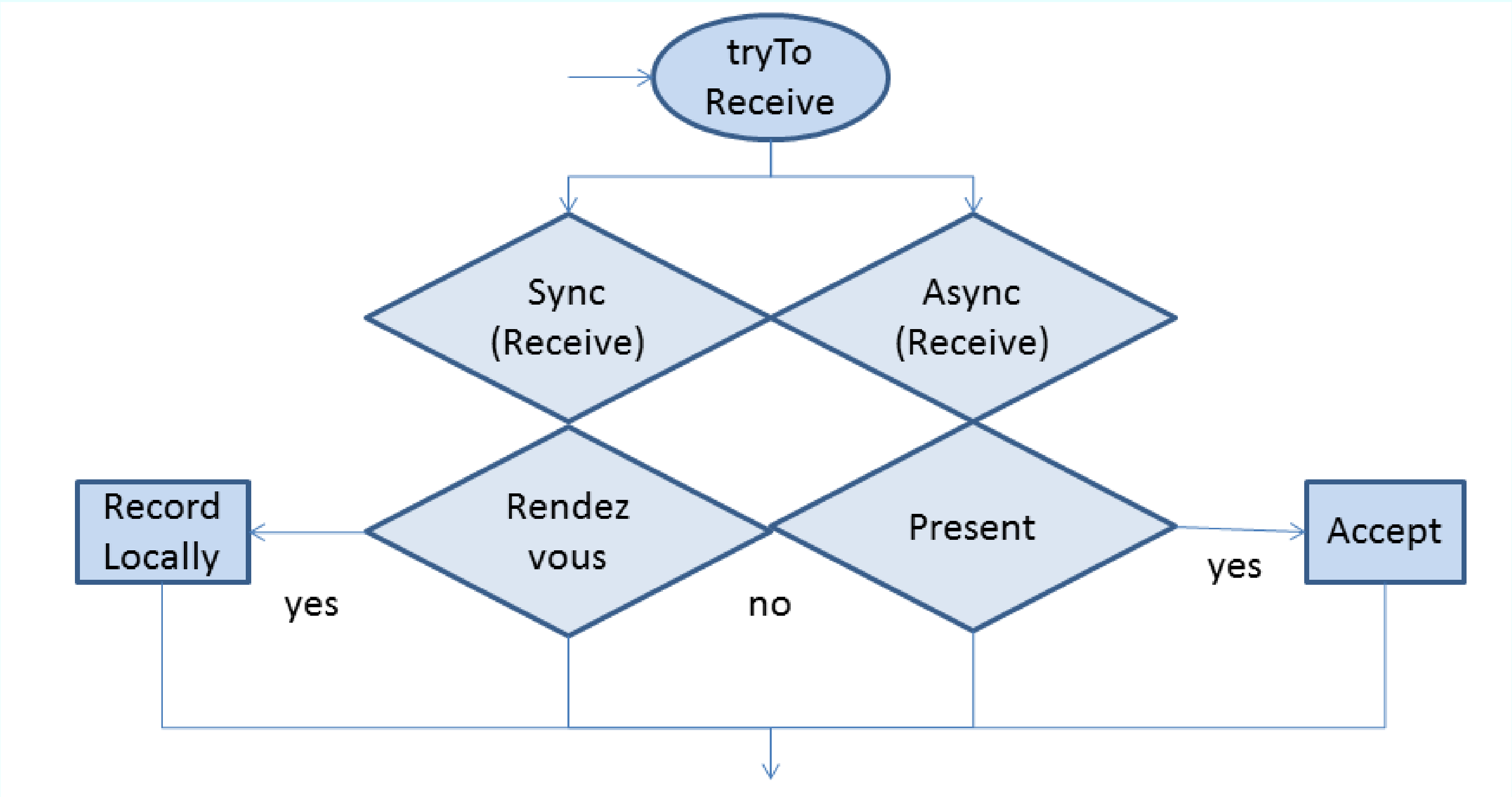

- PREPAREMSG: abstract interface to data handling (message content)

# The component TryToSend




inputPool, if accessible, possibly blocked for async PassMsg

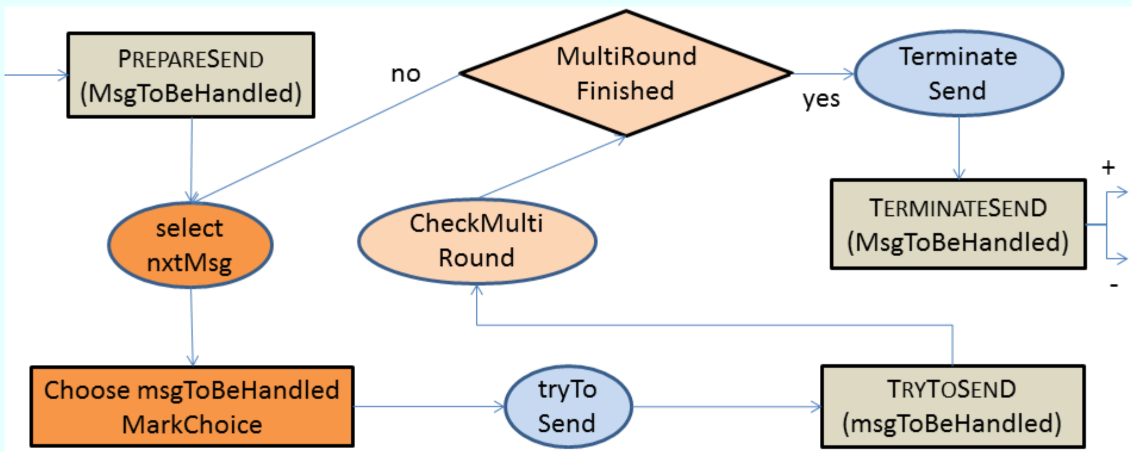
# The component TryToReceive




NB. inputPool configurable to asynchronously/synchronously receive msgToBeHandled of expected kind from expected sender

# Refinement of SingleSend to MULTISEND

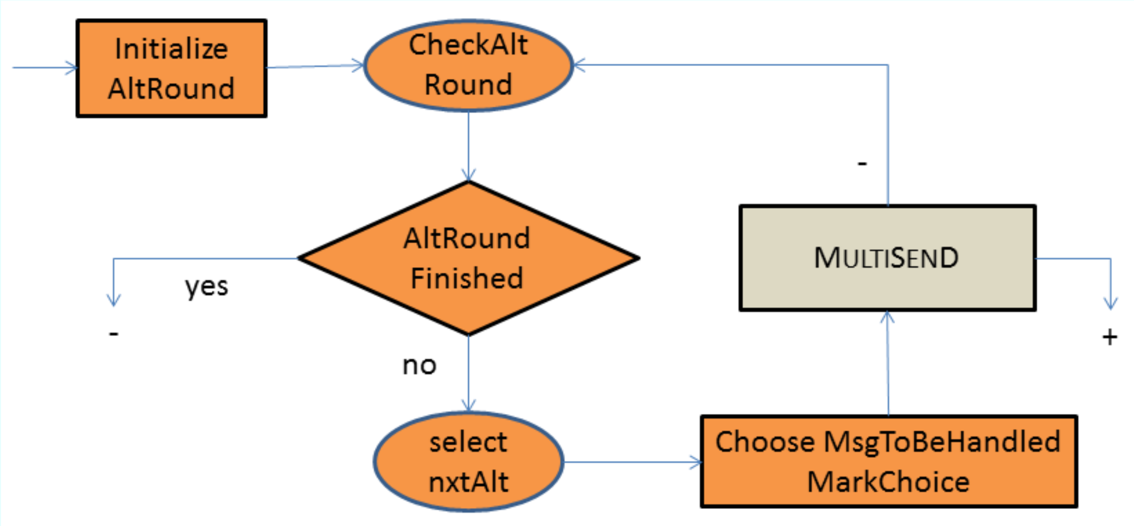
- PREPAREMSG prepares a set $MsgToBeHandled$ of $mult$ many msgs
- **choose** $msgToBeHandled$ which is passed to SingleSend
- TERMINATESEND extended to $CheckMultiRound$ completion




- extension is modular
- refinement is conservative (purely incremental): 'same' behavior as SINGLESEND for singleton sets $MsgToBeHandled$ (where $mult = 1$)

# MultiSend refinement to AlternativeSend TryAlt(*Send*)

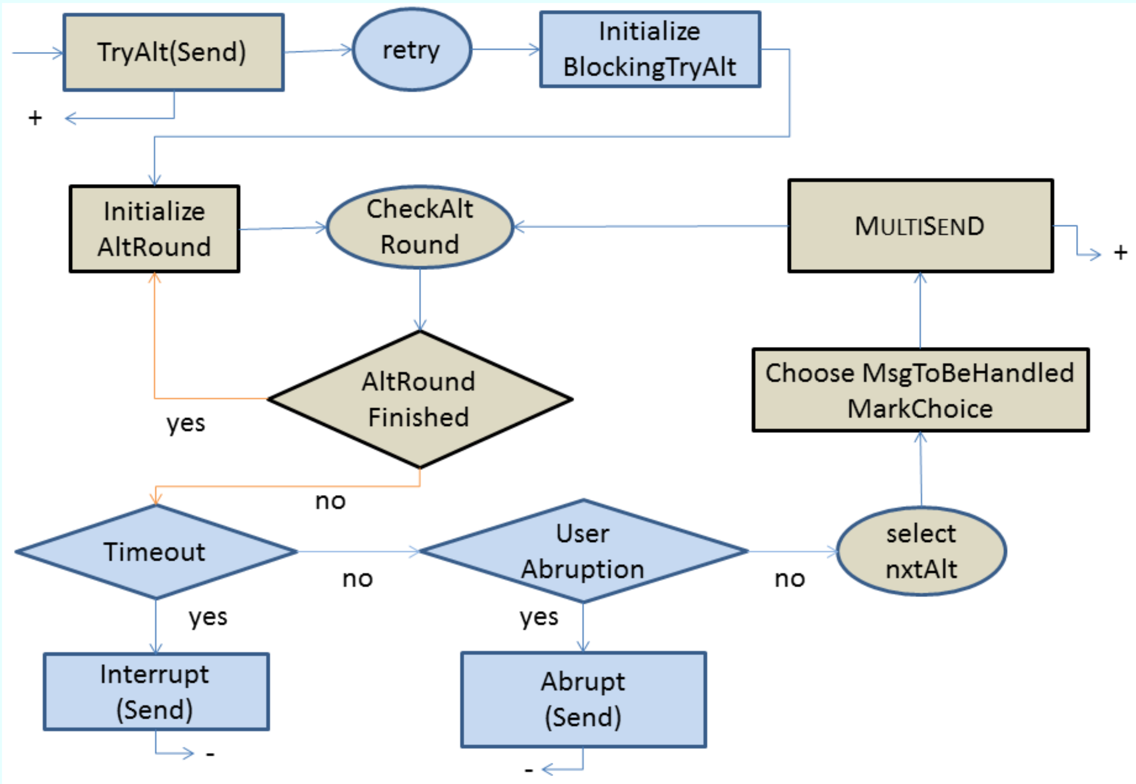
- one by one each alternative *MsgToBeHandled* is selected to be handled by MultiSend




- refinement is conservative: same behavior as MultiSend for singleton sets of alternatives

# Send component $\text{PERFORM}_{subj}(Send)$

- one non-interruptable $\text{TRYALT}(Send)$ round trying all alternatives
- then further time/user interruptable $\text{TRYALT}(Send)$ rounds




Same diagram for Receive with $\text{TRYALT}(Receive)$, MULTIRECEIVE: reusable component-based ASM design

# Conclusion

Close correspondence bw intuitive intended meaning of graphical notations and accurate ASM definitions can be extended to additional (definable) S-BPM concepts like

- *alternative action* control states allowing interleaving
- *design-for-change schemes* to extend SBD by new/exception behavior (model-based mastering of on-the-fly adaptation of running systems)

BPM-ASMs, tailored to model basic concepts of S-BPM (and its tool suite) can serve as ground model descriptions to mediate bw domain expert and sw system designer views of BPs:

- possible because ASM language uses only (semantically well defined) fundamental description as well as reasoning scheme of both natural and scientific languages:

  $$\textbf{if } Condition \textbf{ then } Statement$$

  where $Condition$ (event/property) triggers to-be-performed *action* resp. implies to-be-proved *logical expression* described by $Statement$.

# A BP certification procedure

- build correct models for meaning of (graphical) BP notations
  - define meaning in precise application domain terms
  - define BPM-ASM ground models (*end-user-oriented domain-knowledge-expressing interfaces*) for the meaning
  - validate ground models to 'correctly' represent intended meaning
- provide guaranteed correct BP ground model
  - design BP using above defined (graphical) notations
  - inspect/validate BP design to correctly reflect intentions
- provide guaranteed correct ground model implementation
  - use resulting ground model BPM-ASM of a graphical BP design as precise and complete spec for sw implementation of the BP
  - verify the coding to be correct

Result: implementation is guaranteed (and can be certified) to correctly reflect the meaning the BP expert intended by high-level BPM.

# Degrees of certificate quality

Quality (degree of reliability) of a correctness certificate for a BP is proportional to the quality of:

- the ground model validation, e.g. by model inspection, model checking, model-based testing
- verification of the stepwise refinements used to develop/generate code for an executable version of the BP spec, e.g. by
  - compiling ground model BPM-ASM using a verified compiler
  - providing proof sketches or standard mathematical or machine supported (interactive or fully automated) proofs of (some critical or all) code generating refinement steps

S-BPM approach to BP development offers all the ingredients which allow one to produce certifiably correct industrial BPs

- NB. This is a BP-specific version of Hoare's 'verified software grand challenge'.

# References


- A. Fleischmann, W. Schmidt, C. Stary, S. Obermeier, E. Börger: *Subject-Oriented Business Process Management*.
  – Springer Open Access Book 2012.
- E. Börger and R. Stärk: *Abstract State Machines*. A Method for High-Level System Design and Analysis.
  – Springer-Verlag 2003.

E.Börger: *Approaches to modeling business processes: a critical analysis of BPMN, workflow patterns and YAWL*

- J. Software and Systems Modeling, September 2011 (14 pages) DOI 10.1007/s10270-011-0214-z