Börger Interpreter and the OWL Model Standard

The Börger Interpreter Model for subject-oriented process models is defined in: [1] in the appendix of [2].

* Updated version available: <http://pages.di.unipi.it/borger/Papers/Bpmn/SbpmBookAppendix.pdf>

It uses the formal description mechanism of Abstract State Machines (ASM) [3] to define the algorithm of an Interpreter that will “crawl” through a Subject behavior diagram (SBD) of a process model defined in the, not explicitly named, subject-oriented process modeling language PASS[[1]](#footnote-1) as defined by [4].

Next to the algorithmic definition of the interpreter, a formal/ontologic specification for the passive model structure of PASS has been created within the Subject-Oriented Community (S-BPM community) using the web ontology language (OWL). This is based on the works [5] [6].

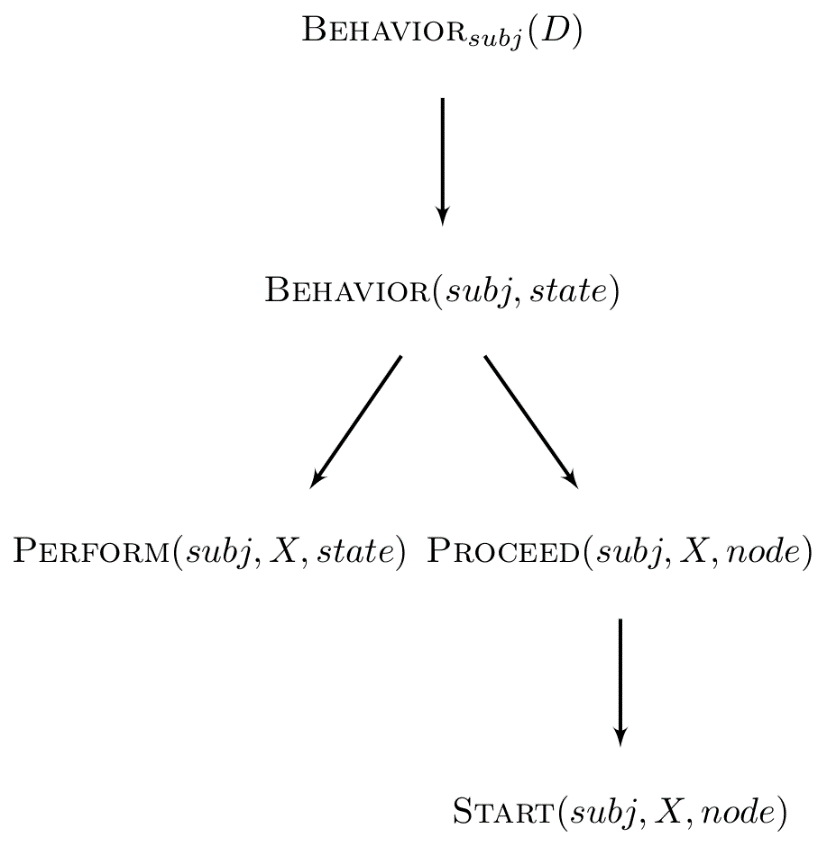
This document matches the active algorithmic definitions of the ASM interpreter with the definitions of the passive PASS model structures.

It describes how ASM Rules and Functions should work on various model elements.

The Interpreter implicitly assumes the following aspects:

* A model is expected to be formally sound, syntactical correct.
* Described is the interpretation of single subject with single behavior (though theoretically the rules could be applied at the same time to multiple behaviors by an according engine)
* The Börger interpreter does describe execution concepts that may not have a directly corresponding counterpart in the models that are to be interpreted

# Core Functions (Börger 2.2 - Semantics of Core Subject Behavior Diagram Transitions)

***Behaviorsubj (D)*** *= {Behavior(subj, node) | node* ∈ *Node(D)}*

***Behavior(subj, state) =***

**if** SID\_state(subj) = state **then**

**if** Completed(subj, service(state), state) **then**

**let** edge = selectEdge({e ∈ OutEdge(state) | ExitCond(e)(subj, state)})

Proceed(subj, service(target(edge)), target(edge))

**else** Perform(subj, service(state), state)

**where**

Proceed(subj, X, node) =

SID\_state(subj) := node

Start(subj, X, node)

Original Main Function/Method/State Machine of the Interpreter. The algorithm or function is called “BEHAVIOR” to define that is that actual behavior of a subject (subj) in a state and that it is implicitly interpreting a Subject Behavior Diagram

The algorithm or function is called “BEHAVIOR” to define that is that actual behavior of a *subject (subj)* in a state and that it is implicitly interpreting a Subject Behavior Diagram

*“We call these states* ***SID\_states*** *(Subject Interaction Diagram states) of the subject in the diagram because they represent the state a subject is in from the point of view of the other subjects it is interacting with in the underlying process”* [1]

* *SID\_State* = the “official” state from outside view
* Not to be confused by a model “state” in an SBD Diagram
* State in the SBD diagram define possible *SID\_States*

The behavior of a single subject *subj* is specified by the *Behaviorsubj (D)* rule, which takes the Subject Behavior Diagram *D* as a parameter. From there on the *Behavior(subj, node)* rule defines how a single node behaves. As long as the service of that node is not *completed* the Perform rule will be called, which is refined for all given services *X*. Once the node is completed the outgoing transition will be determined by *selectEdge*, the *Proceed* rule updates the current *SID\_state* and initializes the new node with the *Start* rule, which also is refined for all services *X*.

## Places

Places are formally also functions or rules, but are used in principle as passive/static storage places.

|  |  |  |
| --- | --- | --- |
| Interpreter Spec | Description | Corresponding OWL-Model Element |
| *SID\_state* | Execution concept – no model representation, Not to be confused by a model “state” in an SBD Diagram. State in the SBD diagram define possible SID\_States. | X - Execution concept – the state the subject is currently in as defined by a **State** in the model |
| *D* | A **D**iagram that is a completely connected SBD | **SubjectBehavior** – under the assumption that it is complete and sound. |
| *node* | A specific element of diagram D   * Every node 1:1 to state | **State** |
| *state* | The current active state of a diagram determined by the *nodes* of Diagram D | **State** |
| *initial state* | The interpreter expects and SBD Graph D to contain exactly one *initial (start)* state and at least one *end state* | **InitialStateOfBehavior** |
| *end state* | **EndState** |
| *edge / outEdge* | “Passive Element” of an edge in an SBD-graph | **Transition** |
| *ExitCondition* | Static Concept that represents a Data condition | **TransitionCondition** |
| *subj* | Identifier for a specific Subject Carrier that may be responsible for multiple Subjects | Execution Concept – ID of a Subject Carrier responsible possible multiple Instances of according to specific **SubjectBehavior** |
| *ExternalSubject* | A representation of a service execution entity outside of the boundaries of the interpreter (The PASS-OWL Standardization community decided on the new Term of Interface Subject to replace the often-misleading older term of External Subject) | Represented in the model with **InterfaceSubject** |
| *subject-SBD /*  *SBDsubject* | Names for completely connected graphs / diagrams representing SBDs | **SubjectBehavior** or rather **SubjectBaseBehavior** as **MacroBehaviors** and **GuardBehaviors** are not covered by Börger |
| *service(state) /*  *service(node)* | Rule/Function that reads/returns the service of function of a given state/node: | Object Property: **hasFunctionSpecification**  (linking **State,** and **FunctionSpecification** -->  (**State hasFunctionSpecification FunctionSpecification**) |
| *function state* | The ASM spec does not itself contain these terms. The description text, however, uses them to describe states with an according service  (e.g. a state in which a (*ComAct = Send)* service is executed is referred to as a *send state*)  Seen from the other side: a **SendState** is a state with *service(state) = Send)*  Both send and receive services are a *ComAct* service.  The *ComAct* service is used to define common rules of these communication services. | **DoState** |
| *send state* | **SendState** |
| *receive state* | **ReceiveState** |
| *ComAct* | Specialized version of Perform-ASM Rule for communication, either send or receive. These rules distinguish internally between send and receive. | **CommunicationAct**swith sub-classes (**ReceiveFunction**  **SendFunction)**  **DefaultFunctionReceive1\_EnvironmentChoice**  **DefaultFunctionReceive2\_AutoReceiveEarliest**  **DefaultFunctionSend** |

## Main Execution/Interpreting Rules

The interpreter ASM Spec has main-function or rules that are being executed while interpreted.

BEHAVIOR(*subj,state*),

PROCEED(*subj,service(state),state*),

PERFORM(*subj,service(state),state*)

START (*subj,X, node)*

These make up the main interpreter algorithm for PASS SBDs and therefore have no corresponding model elements but rather are or contain the instructions of how to interpret a model.

|  |  |  |
| --- | --- | --- |
| Interpreter Spec | Description | Corresponding OWL-Model Element |
| *BEHAVIOR(subj;state)* | Main interpreter ASM-rule/Method | Execution concept |
| *BEHAVIOR(subj;node)* | ASM-Rule to interpret a specific node of Diagram D for a specific subject | Execution concept |
| **Behaviorsubj (D)** | Set of all ASM rules to interprete all nodes/states in a SBD(iagram) D for a given subj (set of all *BEHAVIOR(subj;node))* | Execution concept |
| *PERFORM(subj ; service(state); state)* | The main Perform ASM Rule/Method that prompts an PASS interpreter to execute functions defined for states | **State hasFunctionSpecification FunctionSpecification**  Specialized in: **DoFunction** and. **CommunicationActs** with  **ReceiveFunction**  **SendFunction**  There exist a few default activities: **DefaultFunctionDo1\_EnvoironmentChoice**  **DefaultFunctionDo2\_AutomaticEvaluation** |
| *PERFORM(subj ;ComAct; state)* | ASM-Rule specifying the execution of a Comunication act in an according state) | **CommunicationActs** with  **ReceiveFunction**  **SendFunction**  **DefaultFunctionReceive1\_EnvironmentChoice**  **DefaultFunctionReceive2\_AutoReceiveEarliest**  **DefaultFunctionSend** |

## Functions

Functions return some element. They are activities that can be performed to determine something.

Dynamic functions can be considered as “variables” known from programming languages, they can be read and written.

Static functions are initialized before the execution, they can only be read.

Derived functions "evaluate” other functions, they can only be read. “They may be thought of as a global method with read-only variables” (p. 35)

|  |  |  |
| --- | --- | --- |
| Interpreter Spec | Description | Corresponding OWL-Model Element |
| *SID\_state(subj)* | Dynamic ASM-Function that stores the current *state* of a *subj* | Function that the return state should correspond to/be derived from one of the multiple **State** in an SBD model |
| *OutEdge(state)* *OutEdge(state;i)* | Function that returns the set of outgoing edges of a state or a single specific edge i | **State hasOutgoingTransition Transition**  (input / worked on link / output (Set of Transition)  (linking State with ) |
| *target(edge) /* *target(outEdge)* | Function that returns the follow up state of an outgoing transition (*outEdge* is a special denomination for an *edge* returned by the *OutEdge* -Function) | Object Property: **hasTargetState** (linking **Transition** and **State**-->  **Transition**  **hasTargetState**  **State** |
| *source(edge)* | Function that returns the source state of an *edge* | Object Property: **hasSourceState** (linking **Transition** and **State**-->  **Transition** **hasSourceState** **State** (input / worked on link / output) |
| **Determine Follow up state Mechanic** | | |
| *ExitCond(e)*  *ExitCond(outEdge)*  *ExitCond\_i(e)*  *ExitCond(e)(subj,state)* | Derived Function that evaluates the ExitCondition of a given edge/outgoing edge | Exit conditions in PASS are defined on their corresponding **Transition**s and therefore are called **TransitionCondition**.  **Transition**s have(**hasTransitionCondition)**  (**State -**> **hasOutgoingTransition** --> **Transition** -->  **hasTransitionCondition** --> **TransitionCondition**) |
| *selectEdge* | ASM Function that determines an edged (transition) to follow. | Execution Concept |
| *completed(subj ; service(state); state)* | Function that returns true if the Service of a certain *state* is complete IF the *subj*ect is in that *state* | Execution Concept (connected to: **State,** and **FunctionSpecification**) |
|  | Rule/Function that gives that returns the service of function of a given state: | Execution Concept |

# Extended Concepts (Börger 3 – Refinements for the Semantics of Core Actions)

## Places

|  |  |  |
| --- | --- | --- |
| Interpreter Spec | Description | Corresponding OWL-Model Element |
| *P /InputPool* | The input pool of a subject instance | Execution concept – the model contains of **Subject** that **hasInputPoolConstraints** **InputPoolConstraint**s wich are applied to P |
| *MsgToBeHandled(subj ; state)* | Place where temporarily messages are stored that will be send or received. | Execution concept |

## Input Pool Handling

|  |  |  |
| --- | --- | --- |
| Börger Interpreter Function | Description | Corresponding OWL-Model Element |
| *constraintTable(inputPool )* | Function that Returns the set of all input Pool constrains | Refers to a set of **InputPoolConstraints** of **Subject** that has **hasInputPoolConstraints** – for its Input Pool |
| *sender/receiver* | Identifiers for possible subject instances trying to access an input pool | Execution Concept with evalution relevance for: **MessageSenderTypeConstraint** and **SenderTypeConstraint** |
| *msgType* | (Data) Type of an Instance of a Message | **MessageSpecification**  Can be used to formulate: **MessageTypeConstraint** |
| *selectMsgKind(subj ;state;alt;i)* | ASM Function that determines the message kind (“message type”) to be received in a given receive state. | Execution Concept |
| *{Blocking; DropYoungest; DropOldest; DropIncoming}* | Default Input Pool handling strategies for | **InputPoolContstraintHandlingStrategy**  And their individual defaultinstances: **InputPoolConstraintStrategy-Blocking** **InputPoolConstraintStrategy-DeleteLatest** **InputPoolConstraintStrategy-DeleteOldest** **InputPoolConstraintStrategy-Drop** |
| *P / inputPool* | The actual Input Pool | Execution Concept – can be restricted by **InputPoolConstraint** |
| synchronous communication | Definition for an input pool **constraint set to 0** requiring sender and receiver interpreter to be in the corresponding send and receive states at the same time in order to actually communicate (as messages cannot be passed to an input pool) | |

## Other Functions

|  |  |  |
| --- | --- | --- |
| Interpreter Spec | Description | Corresponding OWL-Model Element |
| *NormalExitCond* | is used internally to “remember” that neither a timeout nor a user cancel have happened, so that the correct exit transition can be taken. | Exit conditions in PASS are defined on their corresponding **Transition**s and therefore are called **TransitionCondition**. Execution Concept: can be set on.  Execution Concept: used to determine the correct exit |
|  |  |  |
| **Timer/Timeout Mechanic:** The evaluation and handling of timeouts is defined (and refined) with several rules and functions. *timeout(state), Timeout(subj , state, timeout(state)), SetTimeoutClock(subj ; state)* are used to evaluate the timeout condition, *Interrupt\_service(state)(subj , state)* is used to define how the corresponding service should be canceled. *TimeoutExitCond* is used internally to “remember” that a timeout happened, so that the correct exit transition can be taken. | | In the model to be interpreted the according aspects are captured by **TimerTransition**s that have (**hasTransitionCondition)** a  **TimerTransitionCondition** containing the date. The *timeout(state)* function should read the information. |
| **User Cancel/Abrupt Mechanic:** The evaluation and handling of user cancels is defined (and refined) with several rules and functions. *UserAbruption(subj, state)* is used to evaluate the user decision, *Abrupt\_service(state)(subj , state)* is used to define how the corresponding service should be abrupted. *AbruptionExitCond* is used internally to “remember” that a user cancel happened, so that the correct exit transition can be taken. | | In PASS models the possibility to arbitrarily cancel the execution of a (receive) function and the possible course of action afterwards may be discerned via a **UserCancelTransition**s |
| *MultiRound / mult(alt) / InitializeMultiRoun /* *ContinueMultiRoundSuccess*  *(among others* | Definition of Functions and ASM rules for interaction between multiple Subjects at once | With the definition of the data properties **hasMaximumSubjectInstanceRestriction**  The **MultiSubject** are actually the standard and **SingleSubject** the special case |
| *AltAction /* *altEntry(D) / altExit(D)* | Rules for the semantics/handling of ChoiceSegements | Handling of **ChoiceSegment** & **ChoiceSegmentPath** |
| *AltBehDgm(altSplit)* *altJoin(altSplit)* |
| *Compulsory(altEntry(D))* and *Compulsory(altExit(D))* |  | Combination of **isOptionalToStartChoiceSegmentPath** and **isOptionalToEndChoiceSegmentPath** data properties |

# Elements Not Covered not by Börger (directly):

This section list elements in the Standard-PASS-OWL standard that are not covered by Börger’s ASM interpreter so far

|  |  |
| --- | --- |
| PASS OWL Model Element |  |
| **InteractionDescribingComponents / Subject / FullySpecifiedSubject/ MultiSubject** | With the exception of InputPool related concepts that belong to the SID of PASS process model, the Börger interpreter is for most parts concerned with the execution of a singular SBD. The specifying and coordinating nature of the SID of a PASS model should have had an influence on the creation of the individual SBDs but therefore is also irrelevant for the ASM interpreter. |
| **FullySpecifiedSubject**  **containsBehavior** min 1 **SubjectBehavior** | The OWL-Model standard envisions a subject to contain possibly multiple behaviors that possess a priority order. This includes **GuardBehaviors**, **MacroBehaviors** and **ExtensionBehaviors/ExtensionLayer** as well as elements to navigate between the different behavior layers such as the **GenericReturnToOriginReference**.  The Börger interpreter cannot execute these Multi-Behavior models.  There is however an ASM Spec extending the Börger Spec that envisions the according execution . [7]  Interpreter ASM Rules/Mechanisms for the Execution of Guards and Macros assumes these specialized model constructs to be part of the singular SBD. That original concept has evolved and been changed in the new PASS-OWL model standard |
| **GuardBehavior** |
| **MacroBehavior** |
| **ExtensionLayer** |
| **GenericReturnToOriginReference** |
| **ReminderTransition / ReminderEventTransitionCondition** | This type time-logic-based transitions did not exist when the original ASM interpreter was conceived. They were added to PASS for the OWL Standard. They can be handled by assuming the existence of an implicit calendar subject that sends an interrupt message (reminder) upon a time condition (e.g. reaching of a calendarial date) has been achieved.  (includes the specialized (**CalendarBasedReminderTransition, TimeBasedReminderTransition** |
| **DataDescribingComponent / DataMappingFunction** | The PASS OWL standard envisions the integration and usage of classic data element (Data Objects) as part of a process model. The Börger Interpreter does not assume the existence of such data elements as part of the model. However, the refinement concept of ASMs could easily been used to integrate according interpretation aspects.  (Includes Elements such as **PayloadDescription** for Messages or **DataMappingFunction** |

# References

|  |  |
| --- | --- |
| [1] | E. Börger, "A Subject-Oriented Interpreter Model for S-BPM," 2012. [Online]. Available: http://www.di.unipi.it/~boerger/Papers/Bpmn/SbpmBookAppendix.pdf. |
| [2] | A. Fleischmann, W. Schmidt, C. Stary, S. Obermeier and E. Börger, Subject-Oriented Business Process Management, Berllin Heidelberg: Springer, 2012. |
| [3] | E. Börger, Abstract State Machines: A Method for High-Level System Design and Analysis, Heidelberg: Springer, 2003. |
| [4] | A. Fleischmann, Distributed systems : software design and implementation, Berlin, Heidelberg: Springer, 1994. |
| [5] | M. Elstermann, Proposal for Using Semantic Technologies as a Means to Store and Exchange Subject-Oriented Process Models, Darmstadt, Germany, 2017. |
| [6] | M. Elstermann and F. Krenn, The Semantic Exchange Standard for Subject-Oriented Process Models, Linz, Austria: ACM, 2018. |
| [7] | M. Elstermann, D. Seese and A. Fleischmann, "Using the Arbitrator Pattern for Dynamik Process-Instance Extensioin in a Workflow Managment System," in *Abstract State Machines, Alloy, B, V.D.M., and Z: Third International Conference, A.B.Z. 2012*, Pisa, Italy, 2012. |

1. Parallel Activity Specification Schema [↑](#footnote-ref-1)