Semantic Investigation of a Control-Flow Subset of BPMN 2.0

Eneia Nicolae Todoran
Department of Computer Science
Technical University of Cluj-Napoca
Cluj-Napoca, Romania
Email: Eneia.Todoran@cs.utcluj.ro

Paulina Mitrea
Department of Computer Science
Technical University of Cluj-Napoca
Cluj-Napoca, Romania
Email: Paulina.Mitrea@cs.utcluj.ro

***Abstract*—Business Process Model and Notation (BPMN), now at version 2.0.2, provides a standard graphical representation for specifying business processes. In this paper we report on the first stage of a semantic investigation of BPMN, using methods in the tradition of programming languages semantics. We consider a control-flow subset of BPMN and an execution architecture based on an intermediate language that we name $\mathcal{L}_{BPMN}$. The execution architecture comprises two main components: a translator which takes as input a BPMN model and generates $\mathcal{L}_{BPMN}$ code, and an interpreter for $\mathcal{L}_{BPMN}$. $\mathcal{L}_{BPMN}$ is a process oriented imperative language providing a combination of concepts, including maximal parallelism and durational activities. We employ the mathematical methodology of metric semantics in designing and relating an operational semantics $\mathcal{O}$ and a denotational semantics $\mathcal{D}$ for $\mathcal{L}_{BPMN}$. We establish the formal relation between $\mathcal{O}$ and $\mathcal{D}$ by using an abstraction operator and a fixed point argument. In this way we prove the correctness of the denotational semantics with respect to the operational semantics. We focus on the semantic investigation of BPMN. We also explain how the operational semantics can serve as a blueprint for an implementation on a client-server architecture.**

I. Introduction

Business Process Model and Notation (BPMN), now at version 2.0.2 [20], provides a standard graphical representation for specifying business processes. BPMN is maintained by Object Management Group (OMG).[1] BPMN is aimed to bridge the gap between the business process design and process implementation. Various implementations of BPMN are available currently, and several papers investigate its semantics. The specification [20] describes the semantics of BPMN using natural language. In [17], [18] a control-flow subset of BPMN is investigated with a Z model [16] of BPMN syntax, and using CSP [7], [12] to describe the semantics. Petri nets are used in [4], [15] to describe the semantics of BPMN with focus on control-flow aspects and transactions. In [5] the execution semantics of a control flow subset of BPMN is formalized as graph rewrite rules.

In this paper we report on the first stage of a semantic investigation of BPMN, using methods in the tradition of programming languages semantics, viz. operational semantics and denotational semantics. We consider a control-flow subset of BPMN and an execution architecture based on an intermediate language that we name $\mathcal{L}_{BPMN}$. The execution architecture comprises two main components: a translator which takes as input a BPMN model and generates $\mathcal{L}_{BPMN}$ code, and an interpreter for $\mathcal{L}_{BPMN}$. We design and relate formally an operational and a denotational semantics for $\mathcal{L}_{BPMN}$.

$\mathcal{L}_{BPMN}$ is a process oriented imperative language providing a combination of concepts, including maximal parallelism and durational activities. $\mathcal{L}_{BPMN}$ is designed to capture the semantics of a variety of BPMN control concepts, including:

- Flow objects
  - Events (Delay Timer Intermediate Events)
  - Activities (Tasks, Sub-Processes)
  - Gateways (Exclusive, Inclusive, Parallel)
- Connecting objects
  - Sequence flow

Our work is closer to [17], [18], [19], where CSP is used to describe the semantics of BPMN and for property checking. The papers investigate control flow aspects and emphasize a process oriented approach to system modelling. However, in the semantic investigation of a subset of BPMN our focus is on establishing the correctness of a denotational semantics with respect to a corresponding operational semantics. Why are we interested in two semantics?

The operational semantics is defined in an algorithmic manner, with emphasis on the steps of the computation, and often serves as a blueprint for an implementation. Its definition is based on a *transition relation*, embedded in a deductive system defined in the style of Plotkin's structured operational semantics (SOS) [11]. Though not dealt with in the present paper, one can employ bisimulation semantics [9] to reason about the properties of systems specified in SOS style.

A denotational semantics is a more abstract model. Initially known as Scott-Strachey semantics, the denotational approach is characterized by the *compositionality principle*: the denotational semantics of a composite construction is defined solely based on the denotational semantics of its syntactic constituents. An elegant theory of domains has been developed, which may use order-theoretic structures [6] or metric spaces [1]. Semantic properties can be verified using specific techniques and tools, e.g., Banach's fixed point theorem in the metric approach. The denotational approach is clearly motivated by the desirability to achieve a modular design.

Next, the relation between a denotational semantics and an operational semantics has to be investigated. The natural question is whether the denotational semantics is *correct*

[1] BPMN was initially developed by Business Process Management Initiative (BPMI); at present it is maintained by Object Management Group (OMG), since the two organizations merged in 2005.

with respect to the operational semantics. The denotational semantics is said to be correct with respect to a corresponding operational semantics if whenever the denotational meanings of two language constructs are equal the operational semantics of the two language constructs are also equal in any syntactic context. A formal definition is provided in section II-A.

In this paper we employ the mathematical methodology of metric semantics [1] in designing and relating an operational semantics $\mathcal{O}$ and a denotational semantics $\mathcal{D}$ for $\mathcal{L}_{BPMN}$. The definition of $\mathcal{O}$ is based on a linear time (power) domain $\mathbf{P}$. An element of $\mathbf{P}$ is a (nonempty and compact) collection of sequences of observables (states). The definition of $\mathcal{D}$ is based on a branching time domain $\mathbf{P}_D$. As it is well-known (see, e.g., chapter 17 in [1]), in order to prove the correctness of $\mathcal{D}$ it is sufficient to find an (abstraction) operator $abs : \mathbf{P}_D \to \mathbf{P}$, (which, in general, is not injective) such that: $\mathcal{O} = abs \circ \mathcal{D}$, where $\circ$ is the operator for function composition.

We define such a function abs that takes $\mathbf{P}_D$ processes, which are tree-like structures, as arguments and yields $\mathbf{P}$ collections as results. In this way we establish the correctness of the denotational semantics $\mathcal{D}$ with respect to the operational semantics $\mathcal{O}$ for the language $\mathcal{L}_{BPMN}$. The operational semantics and the denotational semantics are presented in sections IV and V, respectively. The semantic correctness result is presented in Section VI.

A. $\mathcal{L}_{BPMN}$ main concepts

The combination of concepts embodied in $\mathcal{L}_{BPMN}$ can capture various aspects of a business process model specified in BPMN. $\mathcal{L}_{BPMN}$ is a simple imperative language with typical elements x, x_i, that we call *processes*. We represent a BPMN task as a multi-assignment statement in $\mathcal{L}_{BPMN}$, that we also call an (elementary) *activity*. Such a multi-assignment statement can model, e.g., a data-collection screen, implementing a business document manipulation. The effect of a multi-assignment statement becomes visible for parallel processes only after the passage of a given amount of time.

BPMN can be used to specify business processes that are executed in parallel. The standard does not specify an execution order for parallel tasks, hence we assume a model based on *maximal parallelism* (noninterleaving semantics).

Note that any combination of sequential and parallel (fork and join) composition is allowed in BPMN. Parallel Gateways can be used for synchronizing parallel flows, as explained in section 10.5.4 of [20]. A Parallel Gateway will wait for all incoming flows before triggering the flow through its outgoing Sequence Flows. Such a BPMN model can be translated into an $\mathcal{L}_{BPMN}$ program $(x_1 \parallel \ldots \parallel x_n); x$, where the execution of the process x starts only upon the completion of the parallel execution of all processes $x_1, \ldots, x_n$.

B. The time model

Several timed process calculi have been proposed in the literature, which differ on the basis of a number of time-related parameters (see, e.g., [2], and the references provided therein). Activities (actions) can be *durational*, i.e., their execution can take a certain a amount of time, or they can be *durationless* (instantaneous). Time passing can be measured with respect to some previous event, or with respect to the starting time of the system execution. In the first case we speak of *relative time*; in the second case we speak of *absolute time*. Time passing can be governed by a single *global clock*, or, as in distributed systems, by multiple *local clocks* (which elapse independently, although they define a unique notion of global time). The time model can be discrete o continuous. In a *discrete time* model, time is modelled as a monotonically increasing sequence of integers. In a *continuous time* (or dense time) model the times of events are real numbers.

As explained, e.g., in [19], time information and concurrency are essential for an accurate formal description of BPMN. Business processes are lasting and relatively slow activities, compared to the computation speed of a computer. In designing an execution machine for BPMN we assume a uniform, discrete time framework, based on local clocks and durational activities and processes. Clocks are fictitious entities associated with threads implementing Tasks. All the clocks increase at a uniform rate counting time with respect to a fixed global time frame. Also, following [19], we consider a relative timed semantic model for BPMN. In the metric setting that we employ in this paper [1] it is convenient to consider a discrete time model. In the definition of the denotational semantics time units correspond to contraction steps.

C. Implementation architecture

Our focus is on designing and relating a denotational semantics and an operational for a language $\mathcal{L}_{BPMN}$ embodying some core concepts of a subset of BPMN 2.0. However, our operational semantics can also serve as a blueprint for an implementation of a workflow machine that can execute business processes.

There are several machines that can execute business processes described in (a subset of) BPMN 2.0, e.g., jBPM [21], now at version 6.2, released by the Jboss company. In fact, our implementation solution relies on the graphical designer and the model to XML mapping (BPMN model serialization) provided by jBPM 6.0. Our execution machine uses as input the standard XML representation of a BPMN 2.0 model [20]. The execution architecture comprises a translator which takes as input a BPMN model in XML format and generates $\mathcal{L}_{BPMN}$ code, and an interpreter for $\mathcal{L}_{BPMN}$ designed to run on a client server architecture. The BPMN project is stored on the server side as a shared repository, which can be updated by a main process, also running on the server. The BPMN to $\mathcal{L}_{BPMN}$ translator and the implementation of the $\mathcal{L}_{BPMN}$ interpreter are described in more detail in Section VII.

D. Contribution

We present an execution architecture for a control-flow subset of BPMN based on two main components: a translator and an interpreter. The translator takes as input a BPMN model and generates code in an intermediate language that we name $\mathcal{L}_{BPMN}$. The $\mathcal{L}_{BPMN}$ interpreter is designed to execute both sequential and parallel Tasks. $\mathcal{L}_{BPMN}$ combines maximal parallelism with durational activities and processes. We employ the mathematical methodology of metric semantics [1] in designing and relating an operational semantics $\mathcal{O}$ and a denotational semantics $\mathcal{D}$ for a $\mathcal{L}_{BPMN}$. We establish the

formal relation between $\mathcal{O}$ and $\mathcal{D}$ by using an abstraction operator and a fixed point argument. In this way we prove the *correctness* of $\mathcal{D}$ with respect to $\mathcal{O}$. As far as we know, this is the first paper that presents a denotational model and a comparative semantics investigation for the combination of concepts embodied in $\mathcal{L}_{BPMN}$.

II. Mathematical Preliminaries

The notation $(x \in)X$ introduces the set X with typical element x ranging over X.

Let $f \in X \to Y$ be a function; the function $(\, f \mid x \mapsto y\,) : X \to Y$ is defined (for $x, x' \in X, y \in Y$) by: $(\, f \mid x \mapsto y\,)(x') =$ if $x' = x$ then y else $f(x')$. Instead of $(\,(\, f \mid x_1 \mapsto y_1\,) \cdots \mid x_n \mapsto y_n\,)$ we write $(\, f \mid x_1 \mapsto y_1 \mid \cdots \mid x_n \mapsto y_n\,)$.

Let $f : X \to X$ be a function. When $x \in X$ is such that $f(x) = x$, we call x a *fixed point* of f. When this fixed point is unique, we write $x = fix(f)$.

The semantic models given in this paper are defined following the mathematical methodology of metric semantics [1]. More exactly, we work within the mathematical framework of *1-bounded complete metric spaces*. We assume the following notions are known: *metric* and *ultrametric* space, *isometry* (distance preserving bijection between metric spaces, denoted by '$\cong$'), *complete* metric space, and *compact* set. For details, the reader may consult the monograph [1], for instance.

Some metrics are frequently used in metric semantics. For example, if X is any nonempty set, we can define the *discrete metric* $d : X \times X \to [0,1]$ as follows: $d(x,y) =$ if $x = y$ then 0 else 1. (X, d) is a complete ultrametric space. Also, let $(a \in)A$ be a nonempty set, and $A^\infty = A^* \cup A^\omega$, where $A^*(A^\omega)$ is the set of all finite (infinite) sequences over A. A metric over A^∞ can be defined by $d(x,y) = 2^{-\sup\{\, n \mid x[n]=y[n]\,\}}$, where $x[n]$ denotes the prefix of x of length n, in case $length(x) \geq n$, and x otherwise (by convention, $2^{-\infty} = 0$). d is a Baire-like metric, and (A^∞, d) is a complete ultrametric space. For any set A (by a slight abuse) we denote by $\epsilon (\in A^*)$ the *empty sequence* over A. Also, we use '$\cdot$' as a concatenation (and a prefixing) operator over A^∞ sequences. In particular, when $a \in A$ and $x \in A^\infty$, $a \cdot x$ is the sequence obtained by prefixing a to x.

We recall that if (X, d_X), (Y, d_Y) are metric spaces, a function $f{:}X \to Y$ is a *contraction* if $\exists c \in \mathbb{R}$, $0 \leq c < 1$, $\forall x_1, x_2 \in X : d_Y(f(x_1), f(x_2)) \leq c \cdot d_X(x_1, x_2)$. In metric semantics, it is usual to attach a contracting factor $c = \frac{1}{2}$ to each computation step. When $c = 1$ the function f is called *nonexpansive*. In what follows, we denote by $X \stackrel{1}{\to} Y$ the set of all nonexpansive functions from X to Y.

The following theorem is at the core of metric semantics.

Theorem 2.1 (Banach): Let (X, d_X) be a complete metric space. Each contraction $f : X \to X$ has a *unique* fixed point.

Definition 2.2: Let $(X, d_X), (Y, d_Y)$ be (ultra)metric spaces. We define the following metrics over X, $X \to Y$ (function space), $X \times Y$ (Cartesian product), $X + Y$ (disjoint union defined by $X + Y = (\{1\} \times X) \cup (\{2\} \times Y)$), and $\mathcal{P}(X)$ (powerset of X), respectively.

(a) $d_{\frac{1}{2}\cdot X} : X \times X \to [0,1]$
$d_{\frac{1}{2}\cdot X}(x_1, x_2) = \frac{1}{2} \cdot d_X(x_1, x_2)$

(b) $d_{X \to Y} : (X \to Y) \times (X \to Y) \to [0,1]$
$d_{X \to Y}(f_1, f_2) = \sup_{x \in X} d_Y(f_1(x), f_2(x))$

(c) $d_{X \times Y} : (X \times Y) \times (X \times Y) \to [0,1]$
$d_{X \times Y}((x_1, y_1), (x_2, y_2)) =$
$max\{d_X(x_1, x_2), d_Y(y_1, y_2)\}$;

(d) $d_{X+Y} : (X + Y) \times (X + Y) \to [0,1]$
$d_{X+Y}(u, v) =$ if $(u, v \in X)$ then $d_X(u, v)$
else if $(u, v \in Y)$ then $d_Y(u, v)$ else 1

(e) $d_H : \mathcal{P}(X) \times \mathcal{P}(X) \to [0,1]$
$d_H(U, V) = max\{\sup_{u \in U} d(u, V), \sup_{v \in V} d(v, U)\}$
where $d(u,W) = \inf_{w \in W} d(u, w)$ and by convention $\sup \emptyset = 0$ and $\inf \emptyset = 1$; d_H is the *Hausdorff* metric.

We use the abbreviations $\mathcal{P}_{co}(X)$ and $\mathcal{P}_{nco}(X)$ to denote the powerset of *compact* and *non-empty and compact* subsets of X, respectively. Also, we often suppress the metrics part in domain definitions, and write only $\frac{1}{2} \cdot X$ instead of $(X, d_{\frac{1}{2}\cdot X})$.

Remark 2.3: Let $(X, d_X), (Y, d_Y), d_{\frac{1}{2}\cdot X}$, $d_{X \to Y}, d_{X \times Y}$, d_{X+Y} and d_H be as in Definition 2.2. If d_X, d_Y are ultrametrics, then so are $d_{\frac{1}{2}\cdot X}, d_{X \to Y}, d_{X \times Y}, d_{X+Y}$ and d_H. Moreover, if $(X, d_X), (Y, d_Y)$ are complete then $\frac{1}{2} \cdot X$, $X \to Y$, $X \stackrel{1}{\to} Y$, $X \times Y$, $X + Y$, $\mathcal{P}_{co}(X)$ and $\mathcal{P}_{nco}(X)$ with their metrics defined above are also complete metric spaces [1].

A. Semantic correctness

A semantics is a function $\mathcal{M} : \mathcal{L} \to \mathbf{M}$, where $\mathcal{L}$ is a (formal) language and $\mathbf{M}$ is a mathematical domain (of meanings). Let $(x \in)\mathcal{L}$ be a language, C a typical element of a class of syntactic contexts for $\mathcal{L}$, $\mathcal{D} : \mathcal{L} \to \mathbf{D}$ a denotational semantics, and $\mathcal{O} : \mathcal{L} \to \mathbf{O}$ an operational semantics.

Intuitively, a *syntactic context* is a language construct with 'holes'. For example, let $\mathcal{L}$ be a very simple language providing elementary actions a taken from a set $(a \in)A$ and sequential composition written $x_1; x_2$. $\mathcal{L}$ is specified by the little grammar: $x ::= a \mid x; x$. The class of syntactic contexts for $\mathcal{L}$ can be defined as follows: $C ::= (\cdot) \mid a \mid C; C$. For a given context C one denotes by $C(x)$ the result of replacing all occurrences of the 'hole' symbol $(\cdot)$ with x in $\mathbf{C}$. This notation can be defined inductively: $(\cdot)(x) = x$, $a(x) = a$, $(C_1; C_2)(x) = C_1(x); C_2(x)$.

The denotational semantics $\mathcal{D}$ is said to be *correct* with respect to the operational semantics $\mathcal{O}$ in case:

$$\forall x_1, x_2 \in \mathcal{L}\,[\,\mathcal{D}[\![P]\!] = \mathcal{D}[\![Q]\!] \Rightarrow \forall C\,[\,\mathcal{O}[\![C(x_1)]\!] = \mathcal{O}[\![C(x_2)]\!]\,]]$$

Remark 2.4: In order to prove the correctness of $\mathcal{D}$ it is sufficient to find an operator $abs : \mathbf{D} \to \mathbf{O}$ such that: $\mathcal{O} = abs \circ \mathcal{D}$ (see, e.g., [1], chapter 17).

If $\mathcal{D}$ is correct and it is also *complete* with respect to $\mathcal{O}$ then $\mathcal{D}$ is said to be *fully abstract* (we do not need the notion of semantic completeness in this paper). The full-abstraction problem was raised by Robin Milner [8].

III. Syntax of $\mathcal{L}_{BPMN}$ and Informal Explanation

In our semantic investigation the behavior of a BPMN Task is described by its duration and effect. The effect is

modelled in $\mathcal{L}_{BPMN}$ as a multi-assignment statement, defined with the aid of a class of expressions. Also, we use a class of Boolean expressions to model BPMN Gateway conditions. The syntax of $\mathcal{L}_{BPMN}$ is given in Definition 3.1. The basic components are a set $(v \in)V$ of *variables* and a class $(e \in)E$ of *expressions*, whose type(s) we leave unspecified because our focus is on control-flow aspects. Let $(b \in)B$ be a class of *boolean expressions*. We also use $(t \in)T = \mathbb{N}^+$ as the set of (discrete) *time values*, where $\mathbb{N}^+ = \mathbb{N} \setminus \{0\}$, namely the set of natural numbers without 0.

Definition 3.1: ($\mathcal{L}_{BPMN}$ syntax) We define the class $(a \in)A$ of *multi-assignment statements* and the the class $(x \in)X$ of $\mathcal{L}_{BPMN}$ *processes* by:

$$a ::= [] \mid [(v := e,)^* \; v := e]$$

$$x ::= \mathsf{skip} \mid (a, t) \mid \mathsf{if}\ b\ \mathsf{then}\ x\ \mathsf{else}\ x \mid \mathsf{while}\ b\ \mathsf{do}\ x \mid x; x \mid x \parallel x$$

A pair (a, t) is called an (atomic) *activity*.

The semantics of (Boolean) expressions we define in a standard manner. We assume given a set $(\sigma \in)\Sigma = V \to Val$ of *states*, and two valuations $\mathcal{E}[\![\cdot]\!] : E \to \Sigma \to Val$, and $\mathcal{B}[\![\cdot]\!] : E \to \Sigma \to Bool$, where Val is some given set of *values*, and $Bool = \{true, false\}$.

skip is the *inoperative statement*. (a, t) is an $\mathcal{L}_{BPMN}$ *activity*. $a \equiv [v_1 := e_1, \ldots, v_n := e_n]$ is a multi-assignment statement ($n \geq 0$). $\mathcal{L}_{BPMN}$ also provides constructions for conditional statement (if) and while loop (while), sequential composition $(x; x)$ and parallel composition $(x \parallel x)$.

Notation 3.2: In the above paragraph - and everywhere later - we use the symbol '$\equiv$' to denote syntactic identity.

The semantic models of $\mathcal{L}_{BPMN}$ are presented in Section IV and Section V, respectively. They are designed under the following assumptions. There is a shared global repository of (business process) data. Parallel Tasks, implemented as parallel $\mathcal{L}_{BPMN}$ statements, can share data. Parallel Tasks are executed in a maximal parallel manner. However, if two parallel tasks attempt to modify some shared data, the result cannot be predicted. The effect is given by the last task that performs an update operation.

A. BPMN to $\mathcal{L}_{BPMN}$ translator

We consider an implementation architecture for a control flow subset of BPMN 2.0. The architecture comprises a translator component, and an interpreter for $\mathcal{L}_{BPMN}$. The translator takes as input a BPMN model (in XML format) and generates $\mathcal{L}_{BPMN}$ code. Our prototype implementation is developed under the assumption that BPMN Tasks, Events and Gateways can be annotated with labels. Such labels can be attached to or placed inside the shape of the BPMN element, representing its attributes (according to [20], Chapter 7). We use Task annotations to specify the variable names of a multi-assignment $\mathcal{L}_{BPMN}$ statement. Also, we use Gateway annotations to specify the conditions modeling Boolean expressions of an $\mathcal{L}_{BPMN}$ conditional or a loop statement.

The translator component works in several steps: a BPMN to XML serialization, followed by a transalation from XML into an internal graph representation (with loops detection), followed by $\mathcal{L}_{BPMN}$ code (and abstract syntax tree representation) generation. Further details are provided in Section VII.

Our translator, maps any BPMN Task to an $\mathcal{L}_{BPMN}$ activity (a, t). The time parameter t denotes the duration of the Task (at implementation level the duration of a Task is computed as the difference between the completion time and the starting time of the Task). A multi-assignment statement $[v_1 := e_1, \ldots, v_n := e_n]$ can model a business document manipulation (e.g., in the form of a data-collection screen with multiple fields).

$\mathcal{L}_{BPMN}$ provides compositionality at the level of syntax (and semantics), in terms of operators which can combine simple components into more complex ones. $\mathcal{L}_{BPMN}$ can describe complex behaviors assembled in Sub-Processes. However, some features related to BPMN Sub-Processes, such as scoping, are ignored in this paper.

A BPMN Delay Timer Event is translated into an activity $([], t)$, where $[]$ is the empty multi-assignment statement, and t is the parameter of the timer.

For the structured control constructs the translation algorithm follows the patterns presented in [20], chapter 14, where a BPMN to WS-BPEL mapping is presented. A parallel (fork and join) Gateway is translated into a parallel composition of $\mathcal{L}_{BPMN}$ statements. A Sequence Flow is translated into a sequential composition of $\mathcal{L}_{BPMN}$ statements. Conditionals and loops are also handled as in in [20], chapter 14.

IV. Operational Semantics ($\mathcal{O}$)

It is convenient to extend the set of $\mathcal{L}_{BPMN}$ statements with threads. A thread is an activity with a local clock.

Definition 4.1: Let $T_A \subseteq A \times T \times T$ be given by $T_A = \{(a, t, t') \mid t' < t\}$. T_A is a set of triples (a, t, t'), that we call *threads*. The component t' of a thread (a, t, t') behaves as a (discrete time local) *clock*, i.e., a variable whose values range over the non-negative integers and which increases at the same rate as time. Notice that the clock t' of a thread (a, t, t') must satisfy the invariant condition $t' < t$.

We define the set $(r \in)Conf$ of *configurations* by:

$$Conf = X' \cup \{E\}$$

The set $(x \in)X'$ is given by:

$$x ::= \mathsf{skip} \mid (a, t) \mid (a, t, t') \mid \mathsf{if}\ b\ \mathsf{then}\ x\ \mathsf{else}\ x \mid \mathsf{while}\ b\ \mathsf{do}\ x \mid x; x \mid x \parallel x$$

where $(a, t) \in A$ is an activity and $(a, t, t') \in T_A$ is a thread. Obviously, $X \subseteq X', X \neq X'$ (and $X \subseteq Conf$, $X \neq Conf$).

The parameter t gives the duration of an activity (a, t) or a thread (a, t, t'). The thread clock t' must satisfy the invariant condition $t' < t$. Clocks in parallel threads increase at the same uniform rate, counting time with respect to a fixed global time frame.

Notation 4.2: We use the convention that a configuration $E; x$, $E \parallel x$, or $x \parallel E$ is syntactically identified with x. Also, we identify $E \parallel E$ with E. Formally, for any $x \in Conf$,

we have $E; x \equiv x$, $E \parallel x \equiv x$, $x \parallel E \equiv x$ and $E \parallel E = E$. We use the symbol '$\equiv$' to express syntactic identity. Similar conventions are used systematically in [1], as they allow a somewhat more concise formulation of the specification.

We define an operational semantics for $\mathcal{L}_{BPMN}$ by means of a transition relation embedded in a deductive system in the style of Plotkin's structured operational semantics [11].

The definition of the operational semantics of $\mathcal{L}_{BPMN}$ is based on a transition relation $\rightarrow \subseteq (Conf \times \Sigma) \times (Conf \times \Sigma)$. We write $(r, \sigma) \rightarrow (r', \sigma')$ to express that $((r, \sigma), (r', \sigma')) \in \rightarrow$. We use the following notation convention:

$$(r_1, \sigma_1) \nearrow (r_2, \sigma_2) \quad \text{is an abbreviation for} \quad \frac{(r_2, \sigma_2) \rightarrow (r', \sigma')}{(r_1, \sigma_1) \rightarrow (r', \sigma')}$$

Definition 4.3: The transition relation for $\mathcal{L}_{BPMN}$ is the smallest subset of $(Conf \times \Sigma) \times (Conf \times \Sigma)$ satisfying the following axioms and rules:

(A1) $(\ \mathsf{skip}\ , \sigma) \rightarrow (E, \sigma)$

(A2) $((a, 1), \sigma) \rightarrow (E, \varsigma(a, \sigma))$

(A3) $((a, t), \sigma) \rightarrow ((a, t, 1), \sigma) \quad \text{if } t > 1$

(A4) $((a, t, t'), \sigma) \rightarrow (E, \varsigma(a, \sigma)) \quad \text{if } t = t' + 1$

(A5) $((a, t, t'), \sigma) \rightarrow ((a, t, t' + 1), \sigma) \quad \text{if } t > t' + 1$

(R5) $(\ \mathsf{while}\ b\ \mathsf{do}\ x, \sigma) \nearrow$
$(\ \mathsf{if}\ b\ \mathsf{then}\ x;\ \mathsf{while}\ b\ \mathsf{do}\ x\ \mathsf{else}\ \ \mathsf{skip}\ , \sigma)$

(R6) $(\ \mathsf{if}\ b\ \mathsf{then}\ x_1\ \mathsf{else}\ x2, \sigma) \nearrow (x_1, \sigma)$
$\mathsf{if}\ \mathcal{B}[\![b]\!](\sigma) = true$

(R7) $(\ \mathsf{if}\ b\ \mathsf{then}\ x_1\ \mathsf{else}\ x2, \sigma) \nearrow (x_2, \sigma)$
$\mathsf{if}\ \mathcal{B}[\![b]\!](\sigma) = false$

(R8) $$\frac{(x_1, \sigma) \rightarrow (r_1, \sigma_1)}{(x_1; x_2, \sigma) \rightarrow (r_1; x_2, \sigma_1)}$$

(R9) $$\frac{(x_1, \sigma) \rightarrow (r_1, \sigma_1) \quad (x_2, \sigma_1) \rightarrow (r_2, \sigma_2)}{(x_1 \parallel x_2, \sigma) \rightarrow (r_1 \parallel r_2, \sigma_2)}$$

(R10) $$\frac{(x_1, \sigma_1) \rightarrow (r_1, \sigma_2) \quad (x_2, \sigma) \rightarrow (r_2, \sigma_1)}{(x_1 \parallel x_2, \sigma) \rightarrow (r_1 \parallel r_2, \sigma_2)}$$

where $\varsigma([], \sigma) = \sigma$, and $\varsigma([v_1 := e_1, \ldots, v_n := e_n], \sigma) = (\ \sigma \mid v_1 \mapsto \mathcal{E}[\![e_1]\!](\sigma) \mid \cdots \mid v_n \mapsto \mathcal{E}[\![e_n]\!](\sigma)\ )$.

According to axioms (A2) and (A3), an activity $(a, 1)$ with duration 1 is executed as a durationless activity (an instant action) that immediately updates the state; if $t > 1$ then a thread $(a, t, 1)$ is created, with a clock initialized to 1.

Acording to axioms (A4) and(A5), a thread (a, t, t') is executed by incrementing its clock t' until $t = t' + 1$; when $t = t' + 1$ the thread terminates and the state is updated by using the mapping $\varsigma(a, \sigma)$.

The rules (R5)-(R7) for (while) loop and (if) conditional statement are standard. Rule (R8) define the semantics o sequential composition. According to rules (R9) and (R10), parallel $\mathcal{L}_{BPMN}$ statements are executed in a maximally parallel manner. For further explanations concerning the semantics of maximal parallelism in the metric approach, the reader may consult, e.g., [1], chapter 15, section 15.2.

Notice that, when two (or more) $\mathcal{L}_{BPMN}$ parallel components share data, the effect cannot be predicted. The effect is given by the last component (parallel Task) that performs an update operation. Also, notice that in $\mathcal{L}_{BPMN}$ local clocks in all parallel components increase at the same time rate.

Example 4.4: Let $x = ([v := 10], 2) \parallel ([v := 20], 2)$, and $x_1 = ([v := 10], 2, 1) \parallel ([v := 20], 2, 1)$. Notice that $x, x_1 \in Conf$, $x \in X$ (but $x_1 \notin X$). Let $\sigma \in \Sigma$, and let $\sigma_1 = (\ \sigma \mid v \mapsto 10\ )$, $\sigma_2 = (\ \sigma \mid v \mapsto 20\ )$. One can check that: $(x, \sigma) \rightarrow (x_1, \sigma)$. Also, $(x_1, \sigma) \rightarrow (E, \sigma_1)$ and $(x_1, \sigma) \rightarrow (E, \sigma_2)$.

A semantic description does not specify an implementation, but it can suggest one. Based on the operational semantics presented in this section we design a distributed implementation of $\mathcal{L}_{BPMN}$ that can be deployed on a client-server system. The execution architecture comprises a main process designed to run on the server and a collection of parallel threads. The main process can update a shared (business) data repository, and is responsible for the creation and coordination of threads that can be executed on different machines (to serve business tasks). The main process communicates with the threads and performs update operations (on the shared data repository) upon threads completion. Threads are created and scheduled according to the $\mathcal{L}_{BPMN}$ specification. For example, in order to run the $\mathcal{L}_{BPMN}$ program $((a_1, t_1) \parallel (a_2, t_2)); x$, the main process will create two threads that execute (in parallel) (a_1, t_1) and (a_2, t_2). The rest of the program, namely x, is temporarily suspended. The execution of x begins only after the completion of the parallel execution of both (a_1, t_1) and (a_2, t_2).

Remark 4.5: The scheduling structure of the semantic interpreter can be made explicit by using continuation semantics for concurrency (CSC) [3], [13], [14]. One can use CSC in designing a semantic interpreter for $\mathcal{L}_{BPMN}$ where a continuation is a collection of threads combined in parallel with a main process. The main process is essentially an element of the type X', in which threads are replaced by identifiers used as thread references. In our model each thread implements a BPMN Task. At any moment at most one instance of a Task is active. At implementation level, Task identifiers (generated automatically in the XML representation of a BPMN model) can be used as thread identifiers.

In proofs we use structural induction and induction based on the following complexity measure (the mapping c is well-defined; see [1], chapters 1 and 7).

Definition 4.6: We define $c : Conf \rightarrow \mathbb{N}$ by:

$$c(E) = 0 \qquad c(\ \mathsf{skip}\ ) = c(a, t) = c(a, t, t') = 1$$

$$c(\ \mathsf{if}\ b\ \mathsf{then}\ x_1\ \mathsf{else}\ x_2) = 1 + max\{c(x_1), c(x_2)\}$$

$$c(\ \mathsf{while}\ b\ \mathsf{do}\ x) = 1 + c(\ \mathsf{if}\ b\ \mathsf{then}\ (x;\ \mathsf{while}\ b\ \mathsf{do}\ x)\ \mathsf{else}\ \ \mathsf{skip}\ )$$

$$c(x_1; x_2) = 1 + c(x_1)$$

$$c(x_1 \parallel x_2) = 1 + max\{c(x_1), c(x_2)\}$$

Definition 4.7: (Operational semantics $\mathcal{O}[\![\cdot]\!]$ for $\mathcal{L}_{BPMN}$) Let $(S \in) Sem_O = (Conf \times \Sigma) \rightarrow \mathcal{P}_{nco}(\Sigma^\infty)$.[2] Let $\Psi : Sem_O \rightarrow Sem_O$ be given by:

[2]The construction Σ^∞ was introduced in Section II.

$$\Psi(S)(E,\sigma) = \{\epsilon\}$$
$$\Psi(S)(x,\sigma) = \bigcup\{\sigma' \cdot S(r,\sigma') \mid (x,\sigma) \to (r,\sigma')\}$$

where we use the notation $\sigma \cdot p = \{\sigma \cdot q \mid q \in p\}$, for any $p \in \mathcal{P}_{nco}(\Sigma^\infty)$. We put $\mathcal{O}_L = fix(\Psi)$.

Let $\mathbf{P} = \Sigma \to \mathcal{P}_{nco}(\Sigma^\infty)$. We define $\mathcal{O}[\![\cdot]\!] : X \to \mathbf{P}$ by $\mathcal{O}[\![x]\!](\sigma) = \mathcal{O}_L(x,\sigma)$.

Remark 4.8: The set $S(r,\sigma) = \{(r',\sigma') \mid (r,\sigma) \to (r',\sigma')\}$ is finite for any $(r,\sigma) \in Conf \times \Sigma$, hence the transition system induced by the transition relation $\to$ is finitely branching. This fact can be proved by an easy induction on $c(r)$, for any $r \in Conf$. It is a standard result in metric semantics that a finitely branching transition system gives rise to a compact operational semantics [1]. Ψ is a contraction (hence it has a *unique* fixed point) in particular, due to the "$\sigma \cdot \ldots$"-step in its definition.

Example 4.9: Let Let $x \in X$, $\sigma, \sigma_1, \sigma_2 \in \Sigma$ be as in Example 4.4. We have: $\mathcal{O}[\![x]\!](\sigma) = \{\sigma\sigma_1, \sigma\sigma_2\}$.

V. Denotational Semantics ($\mathcal{D}$)

We present a denotational model $\mathcal{D}[\![\cdot]\!]$ based on a branching domain $\mathbf{P}_D$. $\mathbf{P}_D$ is the (unique) metric domain [1] satisfying:

$$\mathbf{P}_D \cong \{p_\epsilon\} + (\Sigma \to \mathcal{P}_{nco}(\Sigma \times \frac{1}{2} \cdot \mathbf{P}_D))$$

Definition 5.1: Let $(\phi \in)Op = (\mathbf{P}_D \times \mathbf{P}_D) \stackrel{1}{\to} \mathbf{P}_D$. Let $\Omega_;, \Omega_\parallel : Op \to Op$, be given by:

$$\Omega_;(\phi)(p_\epsilon, p) = p$$
$$\Omega_;(\phi)(p_1,p_2) =$$
$$\lambda\sigma.\{(\sigma_1', \phi(p_1', p_2)) \mid (\sigma_1', p_1') \in p_1(\sigma)\}, \text{ if } p_1 \neq p_\epsilon$$
$$\Omega_\parallel(\phi)(p_\epsilon, p) = \Omega_\parallel(p, p_\epsilon) = p$$
$$\Omega_\parallel(\phi)(p_1,p_2) =$$
$$\lambda\sigma.(\{(\sigma_2, \phi(p_1', p_2')) \mid (\sigma_1, p_1') \in p_1(\sigma), (\sigma_2, p_2') \in p_2(\sigma_1)\} \cup \{(\sigma_2, \phi(p_1', p_2')) \mid (\sigma_1, p_2') \in p_2(\sigma), (\sigma_2, p_1') \in p_1(\sigma_1)\})$$
$$\text{if } p_1 \neq p_\epsilon, p_2 \neq p_\epsilon$$

We define $; = fix(\Omega_;)$, $\parallel = fix(\Omega_\parallel)$. $\Omega_;, \Omega_\parallel$ are contractions, essentially because the occurrences of ϕ in the right-hand sides of the equations are stored in the space $\frac{1}{2} \cdot \mathbf{P}_D$; in [1] various similar operators are defined in this way.

Let $\theta : T_A \to \mathbf{P}_D$ be given (by induction on $(t - t')$):

$$\theta(a,t,t') = \lambda\sigma.\{(\varsigma(a,\sigma), p_\epsilon)\} \text{ if } t = t' + 1$$
$$\theta(a,t,t') = \lambda\sigma.\{(\sigma, \theta(a,t,t'+1))\} \text{ if } t > t' + 1$$

Definition 5.2: (Denotational semantics $\mathcal{D}[\![\cdot]\!]$ for $\mathcal{L}_{BPMN}$) We define $\mathcal{D}[\![\cdot]\!] : X \to \mathbf{P}_D$ by:

$$\mathcal{D}[\![\text{ skip }]\!] = \lambda\sigma.\{(\sigma, p_\epsilon)\}$$
$$\mathcal{D}[\![(a,t)]\!] = \text{ if } (t=1) \text{ then } \lambda\sigma.\{(\varsigma(a,\sigma), p_\epsilon)\} \text{ else } \lambda\sigma.\{(\sigma, \theta(a,t,1))\}$$
$$\mathcal{D}[\![\text{ while } b \text{ do } x]\!] = fix(W)$$
$$\mathcal{D}[\![\text{ if } b \text{ then } x_1 \text{ else } x_2]\!] = \lambda\sigma. \begin{cases} \mathcal{D}[\![x_1]\!](\sigma) & \text{if } \mathcal{B}[\![b]\!](\sigma) \\ \mathcal{D}[\![x_2]\!](\sigma) & \text{if } \neg(\mathcal{B}[\![b]\!](\sigma)) \end{cases}$$
$$\mathcal{D}[\![x_1; x_2]\!] = \mathcal{D}[\![x_1]\!]; \mathcal{D}[\![x_2]\!]$$
$$\mathcal{D}[\![x_1 \parallel x_2]\!] = \mathcal{D}[\![x_1]\!] \parallel \mathcal{D}[\![x_2]\!]$$

where $W \in \mathbf{P}_D \to \mathbf{P}_D$,

$$W = \lambda p.\lambda\sigma. \text{ if } \mathcal{B}[\![b]\!](\sigma) \text{ then } (\mathcal{D}[\![x]\!]; p)(\sigma) \text{ else } \{(\sigma, p_\epsilon)\}$$

By an easy structural induction on $x \in X$, one can check that $\mathcal{D}[\![x]\!] \neq p_\epsilon$, for any $x \in X$. Also, one can check that $d(p; p_1, p; p_2) \leq \frac{1}{2} \cdot d(p_1, p_2)$, for any $p_1, p_2 \in \mathbf{P}_D$ when $p \neq p_\epsilon$. Hence the higher order mapping W is a contraction, and has a *unique* fixed point.

Example 5.3: $\mathcal{D}[\![([v := 10], 2) \parallel ([v := 20], 2)]\!] = \lambda\sigma.\{(\sigma, \lambda\sigma'.\{((\sigma' \mid v \mapsto 10), p_\epsilon), ((\sigma' \mid v \mapsto 20), p_\epsilon)\})\}$. See also Example 4.9.

Remark 5.4: It is easy to check that $\mathcal{D}[\![\text{ while } b \text{ do } x]\!] = \mathcal{D}[\![\text{ if } b \text{ then } (x; \text{ while } b \text{ do } x) \text{ else skip }]\!]$.

VI. Relation between Denotational Semantics and Operational Semantics

We use different semantic domains for $\mathcal{O}[\![\cdot]\!]$ and $\mathcal{D}[\![\cdot]\!]$, hence we cannot expect that $\mathcal{O}[\![x]\!] = \mathcal{D}[\![x]\!]$, on $\mathcal{L}_{BPMN}$. In order to establish the relation between $\mathcal{O}$ and $\mathcal{D}$ we introduce an intermediate semantics $I : Conf \to \mathbf{P}_D$, a branching time operational semantics $\mathcal{O}_D : Conf \to \mathbf{P}_D$, and an abstraction operator $abs : \mathbf{P}_D \stackrel{1}{\to} \mathbf{P}$. By an appeal to Banach's Theorem 2.1, we prove that $\mathcal{O} = abs \circ \mathcal{D}$. We conclude that $\mathcal{D}$ is *correct* with respect to $\mathcal{O}$.

Definition 6.1: We define $\mathcal{I} : Conf \to \mathbf{P}_D$ by:

$$\mathcal{I}(E) = p_\epsilon$$
$$\mathcal{I}(\text{ skip }) = \lambda\sigma.\{(\sigma, p_\epsilon)\}$$
$$\mathcal{I}(a,t) = \text{ if } (t=1) \text{ then } \lambda\sigma.\{(\varsigma(a,\sigma), p_\epsilon)\} \text{ else } \lambda\sigma.\{(\sigma, \theta(a,t,1))\}$$
$$\mathcal{I}(a,t,t') = \theta(a,t,t')$$
$$\mathcal{I}(\text{ while } b \text{ do } x) = fix(W')$$
$$\mathcal{I}(\text{ if } b \text{ then } x_1 \text{ else } x_2) = \lambda\sigma. \begin{cases} \mathcal{I}(x_1)(\sigma) & \text{if } \mathcal{B}[\![b]\!](\sigma) \\ \mathcal{I}(x_2)(\sigma) & \text{if } \neg(\mathcal{B}[\![b]\!](\sigma)) \end{cases}$$
$$\mathcal{I}(x_1; x_2) = \mathcal{I}(x_1); \mathcal{I}(x_2)$$
$$\mathcal{I}(x_1 \parallel x_2) = \mathcal{I}(x_1) \parallel \mathcal{I}(x_2)$$

where $W' \in \mathbf{P}_D \to \mathbf{P}_D$,

$$W' = \lambda p.\lambda\sigma. \text{ if } \mathcal{B}[\![b]\!](\sigma) \text{ then } (\mathcal{I}(x); p)(\sigma) \text{ else } \{(\sigma, p_\epsilon)\}$$

Notice that $\mathcal{I}(x) \neq p_\epsilon, \forall x \in X'$; this property follows by an easy induction on the structure of $x \in X'$ (recall that $Conf = \{E\} \cup X'$). Also, the higher order mapping W' is a contraction, and has a *unique* fixed point.

Remark 6.2: It is easy to check the following properties:

(a) $\mathcal{I}(\text{ while } b \text{ do } x) =$

$\mathcal{I}(\ \mathsf{if}\ b\ \mathsf{then}\ (x;\ \mathsf{while}\ b\ \mathsf{do}\ x)\ \mathsf{else}\ \ \mathsf{skip}\ ),$

(b) $\mathcal{I}(r_1 \parallel r_2) = \mathcal{I}(r_1) \parallel \mathcal{I}(r_2)$, for any $r_1, r_2 \in Conf$.

Lemma 6.3: $\mathcal{D}[\![x]\!] = \mathcal{I}(x)$, for any $x \in X$.

Proof: Easy structural induction on $x \in X$. ■

Definition 6.4: Let $(S \in)Sem_D = Conf \to \mathbf{P}_D$. We put $\mathcal{O}_D = fix(\Psi_D)$, where $\Psi_D : Sem_D \to Sem_D$ is given by:

$$\Psi_D(S)(E) = p_\epsilon$$
$$\Psi_D(S)(x) = \lambda\sigma.\{(\sigma', S(r)) \mid (x,\sigma) \to (r,\sigma')\}$$

Lemma 6.5: $\mathcal{I} = fix(\Psi_D)$.

Proof: We show that $\mathcal{I}(r) = \Psi_D(\mathcal{I})(r)$, $\forall r \in Conf$. We proceed by induction on $c(r)$. Three subcases.

Case $r \equiv\ \mathsf{while}\ b\ \mathsf{do}\ x$.

$\Psi_D(\mathcal{I})(\ \mathsf{while}\ b\ \mathsf{do}\ x)$

$= \Psi_D(\mathcal{I})(\ \mathsf{if}\ b\ \mathsf{then}\ (x;\ \mathsf{while}\ b\ \mathsf{do}\ x)\ \mathsf{else}\ \ \mathsf{skip}\ )$

[Induction hypothesis]

$= \mathcal{I}(\ \mathsf{if}\ b\ \mathsf{then}\ (x;\ \mathsf{while}\ b\ \mathsf{do}\ x)\ \mathsf{else}\ \ \mathsf{skip}\ )$

[Remark 6.2(a)]

$= \mathcal{I}(\ \mathsf{while}\ b\ \mathsf{do}\ x)$

Case $r \equiv x_1 \parallel x_2$.

$\Psi_D(\mathcal{I})(x_1 \parallel x_2) =$

$= \lambda\sigma.\{(\sigma', \mathcal{I}(r)) \mid (x_1 \parallel x_2, \sigma) \to (r, \sigma')\}$

[Remark 6.2(b)]

$= \lambda\sigma.(\{(\sigma_2, \mathcal{I}(r_1) \parallel \mathcal{I}(r_2)) \mid$
$(x_1,\sigma) \to (r_1,\sigma_1), (x_2,\sigma_1) \to (r_2,\sigma_2)\} \cup$
$\{(\sigma_2, \mathcal{I}(r_1) \parallel \mathcal{I}(r_2)) \mid$
$(x_2,\sigma) \to (r_2,\sigma_1), (x_1,\sigma_1) \to (r_1,\sigma_2)\})$

$= \Psi_D(\mathcal{I})(x_1) \parallel \Psi_D(\mathcal{I})(x_2)$ [Induction hyp.]

$= \mathcal{I}(x_1) \parallel \mathcal{I}(x_2) = \mathcal{I}(x_1 \parallel x2)$

■

Definition 6.6: Let $(\phi \in)Op = \mathbf{P}_D \xrightarrow{1} \mathbf{P}$. The mapping $\Omega_{abs} : Op \to Op$ is defined as follows:

$$\Omega_{abs}(\phi)(p_\epsilon) = \lambda\sigma.\{\epsilon\}$$
$$\Omega_{abs}(\phi)(p) = \lambda\sigma.\bigcup\{\sigma' \cdot \phi(p')(\sigma') \mid (\sigma', p') \in p(\sigma)\}$$
$$\text{if } p \neq p_\epsilon$$

We put $abs = fix(\Omega_{abs})$.

Lemma 6.7: $(abs \circ \mathcal{O}_D)(r)(\sigma) = \mathcal{O}_L(r,\sigma)$, for any $r \in Conf, \sigma \in \Sigma$.

Proof: Let $\mathcal{R} : (Conf \times \Sigma) \to \mathcal{P}_{nco}(\Sigma^\infty)$ be given by:

$$\mathcal{R}(r,\sigma) = (abs \circ \mathcal{O}_D)(r)(\sigma) = abs(\mathcal{O}_D(r))(\sigma)$$

It is enough to prove that $\Psi(\mathcal{R})(r,\sigma) = \mathcal{R}(r,\sigma)$, for any $r \in Conf, \sigma \in \Sigma$. We only consider the case when $r \equiv x, x \in X'$.

$\Psi(\mathcal{R})(x,\sigma) = \bigcup\{\sigma' \cdot \mathcal{R}(r',\sigma') \mid (x,\sigma) \to (r',\sigma')\}$

$= \bigcup\{\sigma' \cdot (abs(\mathcal{O}_D(r'))(\sigma') \mid (x,\sigma) \to (r',\sigma')\}$

[Definition 6.4]

$= \bigcup\{\sigma' \cdot (abs(p')(\sigma')) \mid (\sigma',p') \in \mathcal{O}_D(x)(\sigma)\}$

$= (abs(\mathcal{O}_D(x)))(\sigma) = (abs \circ \mathcal{O}_D)(x)(\sigma) = \mathcal{R}(x,\sigma)$

Hence $\mathcal{R} = fix(\Psi)$. By using Banach's Theorem 2.1, $\mathcal{R} = \mathcal{O}_L$, and the desired result is immediate. ■

Theorem 6.8: $\mathcal{O} = abs \circ \mathcal{D}$, on $\mathcal{L}_{BPMN}$.

Proof: For any $x \in X$ we have:

$\mathcal{O}[\![x]\!] = \lambda\sigma.\mathcal{O}_L(x,\sigma)$ [Lemma 6.7]

$= abs(\mathcal{O}_D(x))$ [Lemma 6.5, Theorem 2.1]

$= abs(\mathcal{I}(x))$ [Lemma 6.3]

$= abs(\mathcal{D}[\![x]\!])$

■

As a consequence of Theorem 6.8 and Remark 2.4 we obtain:

Corollary 6.9: $\mathcal{D}$ is *correct* with respect to $\mathcal{O}$.

VII. Implementation Considerations

There are several machines that can execute business processes described in (a subset of) BPMN 2.0, e.g., jBPM 6.2 [21]. Our implementation solution relies on the graphical designer and the model to XML mapping (BPMN model serialization) provided by jBPM 6.0. Our execution machine uses as input the standard XML representation of a BPMN 2.0 model [20]. The execution architecture comprises two main components: a translator that takes as input a BPMN model in XML format and generates $\mathcal{L}_{BPMN}$ code, and an interpreter for $\mathcal{L}_{BPMN}$.

The XML representation is translated into an internal (directed) graph representation of the BPMN model. The various BPMN element types - including Tasks, (Delay Timer) Events, (Conditional and Parallel) Gateways, and Loops (e.g. While and Repeat loops) - are detected at this step. The internal graph representation is then used to generate $\mathcal{L}_{BPMN}$ code and an optimized abstract syntax tree (AST) representation. For the purpose of this prototype implementation we developed a simple recursive translation algorithm that can only deal with structured control constructs. This is sufficient in practice. Our translation algorithm cannot deal with some BPMN models, e.g. with models based on unstructured loops. However, notice that a similar limitation is considered in [20], chapter 14, where a BPMN to WS-BPEL translation is presented.

Next, the $\mathcal{L}_{BPMN}$ program (in AST representation) is executed by an interpreter designed based on the operational semantics of $\mathcal{L}_{BPMN}$ presented in this paper. The execution model comprises a main process and a collection of threads that can be deployed on a client server architecture. Each thread executes a BPMN task. The main process manages the execution order of threads and is designed to run on the server. The data of the BPMN project are also stored on the server as a repository shared by all Tasks.

Our prototype implementation is developed under the assumption that Flow Objects (Tasks, Events and Gateways) can be annotated with labels. Such labels can be attached to or placed inside the shape of the BPMN element, representing its attributes (according to [20], Chapter 7). Task and Gateway attributes are specified in this way graphically at the level of the BPMN model. We use Task annotations to specify the variable names of a multi-assignment $\mathcal{L}_{BPMN}$ statement. Although the aspect is not captured in the formal specification given in this paper, we mention that in the distributed implementation, Task annotations are also used to specify the (physical) machine on which a thread implementing a BPMN Task is to be executed. Also, we use Gateway annotations to specify the conditions modeling Boolean expressions of an $\mathcal{L}_{BPMN}$ conditional or a loop statement.

A. The present state of the implementation

For the purpose of an initial prototype implementation we decided to use Java RMI. The distributed implementation is in progress. It is designed to be deployed on a client-server architecture, with a main process running on the server. As an initial step, we developed a Haskell [10] (sequential) implementation of the operational semantics of $\mathcal{L}_{BPMN}$ presented in Section IV. The Haskell implementation serves as a prototype model for the distributed implementation. For experimentation and testing purposes, in the Haskell implementation execution traces are augmented with time stamps.

The direct Haskell implementation of the operational semantics produces all possible execution traces. It can be used to verify the correctness of the implementation but it is not tractable, hence it can only be tested on toy $\mathcal{L}_{BPMN}$ programs. Therefore we have also developed an implementation where the main process and the parallel threads that implement BPMN tasks are activated in a random manner, thus simulating their parallel execution. This version produces a single execution trace, and the nondeterminism of a (real) distributed implementation is simulated using a (pseudo) random number generator. In general, at subsequent execution in "single trace semantics" the interpreter can produce different results. In "single trace semantics" the Haskell interpreter is reasonably efficient and can be used to test complex $\mathcal{L}_{BPMN}$ programs representing corresponding BPMN models.

VIII. Concluding Remarks and Future Research

We report on the first stage of a semantic investigation of BPMN 2.0 [20]. We consider a control-flow subset of BPMN and an execution architecture based on an intermediate language that we name $\mathcal{L}_{BPMN}$. The execution architecture comprises two main components: a translator which takes as input a BPMN model and generates $\mathcal{L}_{BPMN}$ code, and an interpreter for $\mathcal{L}_{BPMN}$, designed to execute both sequential and parallel Tasks. $\mathcal{L}_{BPMN}$ is a process oriented imperative language providing a combination of concepts, including maximal parallelism and durational activities. By using techniques from metric semantics [1] we design and relate an operational semantics and a denotational semantics for $\mathcal{L}_{BPMN}$. By using an abstraction operator and a fixed point argument we prove the correctness of the denotational semantics with respect to the operational semantics.

We intend to continue the research concerning the semantic foundations of BPMN, by using methods in the tradition of programming language semantics. Our next aim is to design a fully abstract denotational model for the $\mathcal{L}_{BPMN}$ formalism. Also, we will investigate extensions of the intermediate language $\mathcal{L}_{BPMN}$ that can be used in designing execution machines for more comprehensive subsets of BPMN 2.0.

Acknowledgment

This research was supported by ClujIT Cluster POS CCE project "Brained City - Information Technology based Innovative Development of Cluj-Napoca Fully Integrated Urban Ecosystem", SMIS code 49752.

References

[1] J.W. de Bakker, E.P. de Vink, *Control flow semantics,* MIT Press, 1996.

[2] M. Bernardo, F. Corradini, L. Tesei, "Timed process calculi: from durationless actions to durational ones," *Proc. ICTCS*, pp. 21–32, 2014.

[3] G. Ciobanu, E.N. Todoran. "Continuation semantics for asynchronous concurrency," *Fundamenta Informaticae*, vol. 131(3-4), pp. 373–388, 2014.

[4] R.M. Dijkman, M. Dumas, C. Ouyang, "Semantics and analysis of business process models in BPMN," *Information and Software Technology (IST)*, vol. 50(12), pp. 1281–1294, 2008.

[5] R.M. Dijkman, P. van Gorp, "BPMN 2.0 execution semantics formalized as graph rewrite rules," *Lecture Notes in Business Information Processing*, vol. 67, pp. 16–20, 2010.

[6] G. Giertz, D.S. Scott, *Continuous lattices and domains.* Cambridge Univ. Press, 2003.

[7] C. A. R. Hoare, "Communicating sequential processes," *Communications of the ACM*, vol. 21(8), pp. 666–677, 1978.

[8] R. Milner, "Fully abstract models of typed λ-calculi," *Theoretical Computer Science*, vol. 4, pp. 1–22, 1977.

[9] R. Milner, *Communicating and mobile systems: the π-calculus,* Cambridge Univ. Press, 1999.

[10] S. Peyton Jones, J. Hughes (Eds.), Report on the Programming Language Haskell 98: a Non-Strict Purely Functional Language, 1999, available at `http://www.haskell.org/`.

[11] G. Plotkin, "A structural approach to operational semantics," *J. Log. Algebr. Program.*, vol. 60-61, pp. 17–139, 2004.

[12] A.W. Roscoe, *The Theory and Practice of Concurrency,*, Prentice Hall, 1997.

[13] E.N.Todoran, N. Papaspyrou, "Continuations for parallel logic programming," *Proceedings of the 2nd ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP 2000)*, pp. 257–267, 2000.

[14] E.N.Todoran, "Comparative semantics for modern communication abstractions," *Proceedings of 2008 IEEE 4th International Conference on Intelligent Computer Communication and Processing (ICCP 2008)*, pp. 153–161, 2008.

[15] T. Takemura, "Formal semantics and verification of BPMN transaction and compensation," *Proceedings of the IEEE Asia-Pacific Conference on Services Computing*, pp. 284–290, IEEE Computer Press, 2008.

[16] J. C. P. Woodcock, J. Davies, *Using Z: specification, proof and refinement,* Prentice Hall International Series in Computer Science, 1996.

[17] P.Y. Wong, J. Gibbons, "A process semantics for BPMN," *Lecture Notes In Computer Science*, vol. 5256, pp. 355–374, 2008.

[18] P.Y. Wong, J. Gibbons, "Formalisations and applications of BPMN," *Science of Computer Programming*, vol. 76(8), pp. 633–650, 2011.

[19] P.Y. Wong, J. Gibbons, "A relative timed semantics for BPMN," *Electr. Notes in Theoretical Computer Science*, vol. 229(2), pp. 59–75, 2009.

[20] OMG BPMN specification (December 2013), available at `http://www.omg.org/spec/BPMN/2.0.2/PDF/`.

[21] jBPM Version 6.2.0 documentation, available at `http://docs.jboss.org/jbpm/v6.2/userguide/`.