Social Specifications of Business Processes with Azzurra

Fabiano Dalpiaz
Department of Information and Computing Sciences
Utrecht University, the Netherlands

Evellin Cardoso, Giulia Canobbio,
Paolo Giorgini and John Mylopoulos
Department of Information Engineering and Computer Science
University of Trento, Italy

***Abstract*—A business process is first and foremost a social interaction among multiple participants. Business process modeling languages support the description of business processes in operational terms, as collections of interleaved activities conducted by human and software agents. However, such descriptions do not capture adequately the richness of social interaction among participants. To address this deficiency, we propose Azzurra, a specification language for modeling and enacting business processes. Azzurra is founded on social concepts, such as roles, agents and commitments among them, and Azzurra specifications are social models consisting of sets of commitments. As such, Azzurra specifications support flexible executions of business processes, and provide a semantic notion of actor accountability and business process compliance. In this paper, we present syntax and semantics of Azzurra, and we propose algorithms to determine runtime compliance with an Azzurra specification.**

I. Introduction

Specification languages enable to describe systems at an abstract level that hides away implementation detail. As such, they have been found useful for building early models of a system that are readily analyzable to determine its properties before built. Unsurprisingly, there are dozens of specification languages for software (e.g., Z [28] and VDM [16]), hardware, concurrent processes, interfaces, and more.

We aim to develop a specification language for business processes. As such, our language should abstract away operational details, the exact activities to be performed and the control flow. We adopt social concepts as specification primitives. After all, business processes are social phenomena consisting of social interactions among multiple participants. For example, an order fulfillment business process is enacted through social interactions among supply order managers, client liaisons, customers, warehouse managers and shippers.

Existing business process modeling languages (e.g., BPMN [21], BPEL [1]) describe processes in terms of interleaved activities conducted by human and software agents. These languages are operational in that they prescribe the execution of specific activities over time in accordance with a rigid control flow. Van der Aalst rightly observes [31] that it is challenging to specify flexible workflows in such languages. While one can model explicitly all the variants of process, the specification becomes unwieldy and unmanageable, too complex to be extensible or even comprehensible [12], [24]. Declarative workflow languages go a step forward in addressing this problem by expressing only essential temporal precedence constraints between activities [31].

978-1-4673-6630-4/15/$31.00 © 2015 IEEE

This paper takes another step towards specifications that abstract away operational details. Business process modeling languages are grounded on computer system process concepts, rather than social ones. Thus, they require the wrong kind of detail by focusing on how a business process is to be enacted, rather than what it is intended to achieve and who is accountable for it. With an eye on hiding away operational details to enable flexible executions of processes, we adopt the notion of *social commitment* [26] among actors in a business process as the fundamental business process abstraction.

A commitment is a social contract between two actors of the form "An actor commits to another actor to make a condition true if another condition is true". Note that commitments do not specify activities, but rather the conditions that must hold when a commitment is fulfilled. Commitments, like expressions in temporal logic, are declarative. However, unlike expressions in temporal logic, they also constitute a high-level social abstraction that the participants in a business process enact. Commitments explicitly capture the social responsibilities of actors towards each other. Sociologists have known for some time the fundamental role commitments play in understanding social activity [2].

Building on top of commitments [26] and commitment protocols [35], we propose Azzurra, a specification language for business processes that relies upon *social primitives*. Our contributions are as follows:

- We propose an expressive language for specifying business processes as commitment protocols. The language includes business primitives such as delegations, deadlines, and constraints over roles. Azzurra offers a notion of initiation and termination of a protocol, and also supports protocol cross-references.
- We introduce a graphical notation for modeling the main elements of a business process. This notation is supported by a prototype Eclipse-based modeling tool.
- We provide algorithms to determine whether a set of observed events complies with an Azzurra protocol specification. These have been implemented in Java using the Drools rule engine language. Noncompliance can be dealt with by an enactment engine that is able to carry out compensation tactics.
- We present two applications of Azzurra on scenarios from real-world case studies, that clearly compare it against mainstream business process modeling approaches.

The rest of the paper is structured as follows. Section II presents our research baseline. Section III defines syntax and semantics of Azzurra, and Section IV introduces algorithms to detect compliance with an Azzurra specification. Section V describes our implementation of design- and run-time tooling. Section VI evaluates our language through a comparative evaluation that involves two different scenarios. Section VII contrasts Azzurra with related work. Finally, Section VIII presents conclusions and outlines future work.

II. Baseline: commitments and protocols

In order to specify business processes in social terms, we need conceptual primitives for representing social interaction. Our choice is to rely on commitments, which have been studied as a fundamental social primitive in social sciences [2], computer-supported collaborative work [13], and multiagent systems [26]. Commitments are social abstractions, as they carry a social meaning (they are contracts).

A social commitment [26], formally c(x,y,p,q), is a promise with contractual validity made by an agent x (debtor) to another agent y (creditor) that, if proposition p is brought about (antecedent), then proposition q will be brought about (consequent). If p is true ($\top$), the commitment is unconditional; otherwise, it is conditional.

Commitments change when their two agents interact by exchanging messages. Messages constitute *commitment operations*: (i) *creation*: the debtor commits to the creditor that the consequent will be brought about; (ii) *cancellation*: the debtor cancels an existing commitment; (iii) *release*: the creditor releases the debtor from a previous commitment; (iv) *delegation*: the debtor delegates the commitment to a third party; and (v) *assignment*: the creditor assigns its credit to another actor.

Moreover, *declare* operations let an agent inform another that a certain proposition has changed truth value (e.g., the book has been sent). *Declare* operations enable the change of commitment state. A commitment is *detached* when the debtor is informed (through a *declare*) that the antecedent has been brought about, and the commitment becomes unconditional. A commitment is discharged/fulfilled, when the creditor is informed that the consequent has been brought about.

We adopt a version of commitments [19] where antecedent and consequent are expressed in propositional logic extended with a temporal precedence operator "·". Thus, $(\mathsf{p} \wedge \mathsf{q}) \cdot \mathsf{r}$ means that p and q occur (in any order) before r occurs.

Commitments can be abstracted to the class level to define an interaction (business) *protocol* between roles [7], [9]. For instance, given roles R_1 and R_2, a protocol may include a commitment class such as $\mathsf{C}(\mathsf{R}_1, \mathsf{R}_2, \mathsf{P}, \mathsf{Q})$. Thus, an agent playing role R_1 is expected to create instances of this commitment (to some agent playing R_2). The propositions in such commitment will be instantiated too: if P is "Book sent", a possible instance p is "copy 123 of book Dracula sent".

III. Syntax and semantics of Azzurra

We present the Extended BackusNaur Form (EBNF) syntax of Azzurra and its runtime semantics. The syntax is presented in Table I and illustrated in Table II on the fracture treatment scenario from the literature [31]. Fig. 1 shows a graphical notation for visualizing the main elements of an Azzurra specification; the notation can be used via a prototype modeling tool built on top of Eclipse (see Sec. V-A). The semantics is explained textually while describing the EBNF syntax.

Notational conventions. We denote classes with identifiers that have a leading capital letter, and instances with identifiers that have a leading lowercase letter.

Protocol signature (1,3). A protocol (1) has an identifier p_{id} and a set of parameters (3): a "key" variable that is the unique identifier for the instances of that protocol, and a set of agent variables (two or more) associated with specific roles. Protocol designers are responsible for choosing a meaningful key for the protocol. The agent variables indicate those agents that play certain roles when a protocol is instantiated. The semantics of protocol instantiation is explained later in this section.

Example. In the treatment protocol in Table II, the protocol name is Treatment, the key is the hospitalization number hospnr, the agent variables are patient pt and specialist sp.

Protocol body (2). It includes a set of typed agent variables (their type is a role), a set of commitment classes, a set of protocol refinements (optional), and a knowledge base that defines semantic relations between atomic propositions (optional).

Example. In Table II, there are five agent variables, including rc (a rehab centre) and ra (a radiologist), nine commitments (C_1–C_9), and two commitment refinements.

Commitments (5,6). The core of a protocol (5) consists of commitment classes. A commitment in Azzurra (6) extends the semantics presented in our baseline in different ways. First, we introduce the notion of a strong commitment (C^*), where the debtor commits to bring about the consequent only after the antecedent has occurred. Second, given that commitments belong in a specific Azzurra protocol, every state of affairs appearing in the antecedent and consequent of a commitment (e.g., Examined, Diagnosed) has an implicit parameter, i.e., the key of the protocol. This parameter enables relating commitment instances associated with one protocol instance (e.g., examined(121) and diagnosed(234) refer to two different protocol instances, each concerning a specific patient hospitalization). Third, Azzurra enriches the syntax of commitments with triggers and creation deadlines. A trigger—the expression before the $\twoheadrightarrow$ symbol—is an event that triggers a commitment creation. Triggers may have an associated precondition—$[prec]$ in (5)—that indicates that, when the event occurs, the commitment shall be created only if the precondition evaluates to true. A deadline ($\leq time$) specifies that the commitment has to be created within a certain time period after the trigger event fires off. Finally, Azzurra supports two special types of commitments that relate to protocol instantiation and termination:

- *Initial commitments* are created when a protocol is instantiated. Their trigger is "init", an event that occurs when a protocol is instantiated. Debtor and creditor of initial commitments shall be agent variables in the parameters of the protocol. This way, initial commitments are created between couples of agents (debtor and creditor do not refer to unassigned agent variables).

- *Final commitments*: every protocol must contain at least one final commitment. A protocol instance terminates successfully when any of its final commitments is fulfilled, while it terminates unsuccessfully if all final commitments are violated (e.g., cancelled by the debtor). Final commitments are also initial. When a protocol terminates, all debtors of active commitments are released from their responsibility towards the respective creditors.

TABLE II: Azzurra protocol for the fracture treatment scenario

protocol Treatment (**key** hospnr, pt : Patient, sp : Specialist) {
ag-variables: rc : RehabCentre, ra : Radiologist, or : Orthopedist, su : Surgeon, nu : Nurse;
commitments:
init $\twoheadrightarrow$ C_1 : C(sp, pt, $\top$, Examined $\cdot$ Diagnosed $\cdot$ Dehospd) final
NoXRayNeeded $\twoheadrightarrow$ C_2 : C(or, sp, $\top$, SlingMade)
XRayRequested $\twoheadrightarrow$ C_3 : C(ra, sp, $\top$, XRayPerformed)
XRayRequested $\twoheadrightarrow$
C_4 : C^*(sp, ra, XRayPerformed, FractAssessed)
FractAssessed $\twoheadrightarrow$ C_5 : C(or, sp, $\top$, ((Fixated$\oplus$Plastered) $\vee$ fulfil(C_6) $\vee$ SlingMade))
FractAssessed $\twoheadrightarrow_{\leq 2h}$ C_6 : C^*(su, or, SurgeryRequested, Operated)
Operated [$\neg$fused] $\twoheadrightarrow$ C_7 : C(nu, pt, $\top$, RcChosen(rc))
RcChosen(rc) $\twoheadrightarrow$ C_8 : C(rc, pt, $\top$, fulfil-p(RehabGiven, key=hospnr, pat-id=pt, ref-sp=sp))
MedPrescribed(m) $\twoheadrightarrow$ C_9 : C(nu, sp, $\top$, MedApplied(m))
can-deleg-no-resp(C_3)
deadline(C_2, 2h)
protocol refinements:
role-confl(Radiologist,Orthopedist)
kb:
implies(XRayRequested, Diagnosed)
implies(NoXRayNeeded, Diagnosed)
implies(MedPrescribed(m), Diagnosed)
mutExcl(XRayRequested, NoXRayNeeded) }

The agent variables corresponding to debtor and creditor prescribe that:

- if an agent a is assigned to the agent variable, a shall be debtor (or creditor);
- if the agent variable is unassigned, any agent a' can be debtor (or creditor), and a' is assigned to the agent variable by participating in the commitment.

Example. In Table II, C_1 is the only initial and final commitment. The protocol has two agent variable parameters (pt and sp), which are the debtor and the creditor of C_1. When an instance of the protocol is created, with agent frank assigned to sp and agent mel assigned to pt, an instance c_1 of C_1 shall be created with debtor frank and creditor mel. When c_1 is fulfilled (the patient is examined, then diagnosed, and finally dehospitalized), the protocol instance terminates successfully. If c_1 is violated, the protocol terminates unsuccessfully. The triggered commitment C_2 is instantiated only if x-rays are not needed, and it specifies that an or has to commit to sp to make a sling. C_4 shows strong commitments: a specialist commits to assess the fracture only after x-rays have been performed.

Agent variables (2,4). We support agent variables that are unassigned when the protocol is instantiated. They get assigned when an instance of a commitment where they appear is created, and, as an additional effect, the assigned agent adopts the specified role in the protocol instance. Azzurra employs assign-once variables: once an agent is assigned, no other agent can be assigned to that variable.

Example. In Table II, there are agent variables for a rehab centre, a radiologist, an orthopedist, a surgeon, and a nurse. Actual agents will be assigned to these variables as the protocol unfolds, i.e., when commitments are created. For example, an orthopedist will be assigned to or as soon as an instance of C_2 is created.

Commitment refinements (7). A deadline commits the debtor to bring about the consequent within a certain time after the antecedent occurs. The debtor can be authorized to delegate the commitment, either retaining (can-deleg-ret-resp) or releasing (can-deleg-no-resp) her responsibility. The creditor, similarly, can be authorized to assign the commitment, either retaining (can-assign-ret-cred) or releasing (can-assign-no-cred) her credit. The debtor can be authorized to cancel her commitment (can-cancel).

Example. In Table II, the radiologist can delegate instances of C_3, possibly to a colleague, without retaining responsibility. Without such authorization, delegations would correspond to a violation on part of the radiologist.

Protocol refinements (8). They constrain the agents that participate in a protocol instance. The maximum number of concurrent commitments for an agent playing a certain role can be limited (max-per-role), as well as the number of instances of a commitment class that an agent can make (max-of-class). Role conflicts (role-confl) prescribe that an agent cannot play two roles in the same protocol instance. Separation of duties (sep-duties) implies that an agent cannot be debtor in instances of two commitment classes, and it can be restricted to agents playing a specific role (comm-role-confl).

Example. A role-confl refinement specifies that the same agent cannot play both radiologist and orthopedist, because their roles are incompatible in the same protocol instance.

Preconditions, propositions, and triggers (9–15). Azzurra supports different types of preconditions (9) and propositions types (10): atomic (atom), commitment states (cstate), protocol states (pstate), binary operators, and so on. The binary operators (11) are conjunction ($\wedge$), disjunction ($\vee$), exclusive disjunction ($\oplus$), and temporal precedence ($\cdot$). Atomic propositions (15) can be truth ($\top$), falsity ($\bot$), or states of affairs (e.g. FractAssessed). States of affairs may be parametric and, thus, have multiple instances. For example, MedPrescribed(med-id) has an instance for each medication the patient is given. The state of a protocol instance evolves because of the occurrence of events (14), as they trigger new commitment instances and change the state of existing commitment instances. Three event types are supported:

- An atomic proposition becomes true. This includes the occurrence of a state of affairs (e.g., the patient is diagnosed).
- The state of a commitment instance changes (see clause (12) below).

TABLE I: EBNF syntax of Azzurra; terminals in bold, non-terminals in italics

$prot \to$	**protocol** p_{id} ($params$) $\{$	(1)
	[**ag-variables:** $vars$]	
	commitments: $comms\ crefn^*$	
	[**refinements:** $(id : refn)^*$] [**kb:** $domain^+$] $\}$	(2)
$params \to$	**key** v, v **:** $role$ (, v **:** $role)^+$	(3)
$vars \to$	v **:** $role$ (, v **:** $role)^*$;	(4)
$comms \to$	(**init** $\twoheadrightarrow [_{\leq time}]$ $comm$ **final;**$)^+$ (ev [[$prec$]] $\twoheadrightarrow [_{\leq time}]$ $comm$;$)^*$	(5)
$comm \to$	id : **C**[*](v, v, $prop$, $prop$)	(6)
$crefn \to$	**deadline**(id, $time$) $\mid$ **can-deleg-ret-resp**(id) $\mid$ **can-deleg-no-resp**(id) $\mid$ **can-assign-ret-cred**(id) $\mid$ **can-assign-no-cred**(id) $\mid$ **can-cancel**(id)	(7)
$refn \to$	**max-per-role**($role, nr$) $\mid$ **max-of-class**($role$, id, nr) $\mid$ **role-confl**($role$, $role$) **comm-role-confl**($role$, id, id) $\mid$ **sep-duties**(id, id) $\mid$	(8)
$prec \to$	$atom \mid cstate \mid pstate \mid prec\ op\ prec \mid \neg prec \mid$ ($prec$)	(9)
$prop \to$	$atom \mid cstate \mid pstate \mid prop\ op\ prop \mid$ ($prop$)	(10)
$op \to$	$\wedge \mid \vee \mid \oplus \mid \cdot$	(11)
$cstate \to$	**create**(id) $\mid$ **deleg-no-resp**(id [**to** v]) $\mid$ **deleg-ret-resp**(id [**to** v]) $\mid$ **fulfil**(id) $\mid$ **cancel**(id) $\mid$ **expire**(id) $\mid$ **release**(id) $\mid$ **assign-ret-cred**(id [**to** v]) $\mid$ **assign-no-cred**(id [**to** v])	(12)
$pstate \to$	**init-p**(p_{id} (, $v = v)^*$) $\mid$ **fulfil-p**(p_{id} (, $v = v)^*$)	(13)
$ev \to$	**init** $\mid atom \mid cstate \mid pstate$	(14)
$atom \to$	$\top \mid \bot \mid staffairs$ [(v (, $v)^*$)]	(15)
$domain \to$	**implies**($staffairs$, $staffairs$) $\mid$ **mut-excl**($staffairs$(, $staffairs)^+$)	(16)

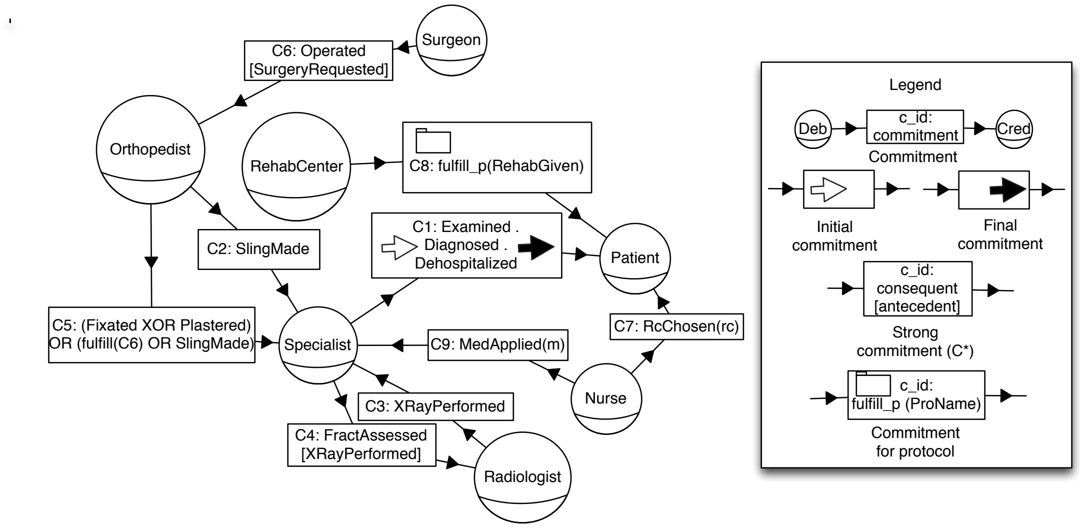


Fig. 1: Graphical representation for the Azzurra protocol in Table II

- The state of another protocol instance changes, i.e., it is instantiated (init-p) or fulfilled (fulfil-p). Optionally, one can specify constraints on the protocol instance parameters, e.g., to impose a certain key or that a specific agent in the current protocol instance shall be assigned to an agent parameter in the referenced protocol.

Example. The consequent of C_5 tells that the commitment is fulfilled if either an instance of Fixated or Plastered occurs (but not both), an instance of C_6 is fulfilled, or an instance of SlingMade occurs. The consequent of C_8 indicates that a successful instance of the protocol RehabGiven is expected, with the constraints that the patient identifier parameter (pat-id) corresponds to the patient in the instance of Treatment, and that the reference specialist (ref-sp) is the specialist who is responsible for the hospitalization of the considered patient.

Commitment states (12). Propositions and may denote that a commitment is in or has changed to a specific state. Given a commitment class id:

- create(id): an instance of id is created;
- deleg-no-resp(id [to v]): an instance of id is delegated (to agent v) without retaining responsibility;
- deleg-ret-resp(id [to v]): an instance of id is delegated (to v); the delegator keeps responsibility;
- fulfil(id): an instance of id is fulfilled;
- cancel(id): an instance of id is canceled;
- expire(id): an instance of id has expired;
- release(id): an instance of id is released;
- assign-ret-cred(id [to v]): an instance of id is assigned (to v) retaining the credit;
- assign-no-cred(id [to v]): id is assigned, but the assignor does not retain the credit.

Knowledge base (16). It specifies semantic relationships, i.e., implications and mutual exclusions, between states of affairs. These relationships belong to the shared vocabulary of the participants in a protocol.

Example. Three states of affairs imply a diagnosis: XRayRequested, NoXRayNeeded, and MedPrescribed. XRayRequested is mutually exclusive with NoXRayNeeded.

IV. Runtime compliance with Azzurra protocols

The semantics of Azzurra specifications enables determining whether the actors participating in a *protocol instance* are compliant with the specification. We assume that the messages that the actors exchange within the context of protocol execution are observable by a monitoring infrastructure. Compliance checking compares the *observed* behavior from occurred events and the *expected* behavior as indicated by the protocol specifications. We present two algorithms that enable determining compliance:

1) Algorithm 1 (EnactProtocols) determines how an event updates the state of existing protocol instances and of the commitment instances therein. We call this activity *enactment* of a protocol. The output constitutes the *expected* behavior.
2) Algorithm 2 (CheckCompliance) checks whether an occurred event violates the specification of a protocol instance. This corresponds to verifying if expected commitments are not created/fulfilled, if disallowed commitment operations are performed, and if protocol constraints (e.g., maximum roles per agent) are violated.

In our algorithms, we assume that the occurring events are associated with a specific protocol instance (there is no ambiguity about which protocol instance they refer to). Events are processed sequentially by dequeuing a first-in first-out queue of events. When the algorithms invoke the ENQUEUE function, an event is added to such queue.

Algorithm 1 enacts a set of protocol instances and the commitments therein contained. The algorithm depends on the type of the processed event *ev*:

- *Protocol instantiation* (lines 1–6): a new protocol instance is created (class, key, and arguments are taken from the event), and added to the protocol instances $\mathcal{P}$ (lines 1–3). An instance of every initial commitment in the protocol specification is created (lines 4–6): debtor and creditor are set by retrieving the agents that are assigned to the agent variables in the commitment class. If specified, a creation deadline is set by adding the creation timeout to the current time (NOW). The following events types refer to the protocol instance that relates to the event (line 7).
- *Commitment trigger* (lines 8–11): commitment instances are created whenever an instance of the trigger event occurs, if the optional precondition holds (the knowledge base of the protocol instance, inferred from all occurred events, entails it).
- *Commitment creation* (lines 12–16): the commitment instance is added to the protocol instance (line 13). If the agent variables for the debtor and the creditor of the corresponding commitment class are still unassigned, their value is assigned to the debtor and the creditor of the commitment instance (lines 14–16).
- *Commitment updates* (lines 17–24): *cancel* and *release* operations imply the removal of the commitment instance from the protocol (lines 17–18). Delegation without retaining responsibility transfers the responsibility to another debtor (line 19), while delegation with retaining responsibility creates a second commitment instance (lines 20–21). Assignments of credit are treated similarly (lines 22–24).

Algorithm 1 Enacting protocol instances based on an occurred event

```
EnactProtocols(Event $ev$, ProtInst [] $\mathcal{P}$)
 1  if $ev = init\text{-}p(p\text{-}id, key, par_1 = ag_1, \ldots, par_n = ag_n)$
 2     then $p \leftarrow$ CreateProtInstance($p\text{-}id, key, ag_1, \ldots, ag_n$)
 3          $\mathcal{P}$.add($p$)
 4          for each $init \twoheadrightarrow_{\leq t} C_i : C(Db, Cd, Ant, Cons) \in p.spec$
 5          do $p$.addCommI($C_i, p$.valOf($Db$), $p$.valOf($Cd$),
 6             $Ant, Cons$, now $+ t$, nil)
 7  ProtInst $p \leftarrow$ getProtInstForEvent($ev$)
 8  if $Ev[Prec] \twoheadrightarrow_{\leq t} C_i : C(Db, Cd, Ant, Cons) \in p.spec \wedge$
 9     $p.kb \vdash prec(p.key)$
10     then $p$.addCommI($C_i, p$.valOf($Db$), $p$.valOf($Cd$),
11          $Ant, Cons$, now $+ t, ev.args$)
12  if $ev = create(db, cd, c)$
13     then $p$.addCommInstance($c$)
14          CommClass $cc \leftarrow p$.classOf($c$)
15          if $p$.valOf($cc.deb$) = nil then $p$.assign($cc.deb, db$)
16          if $p$.valOf($cc.cred$) = nil then $p$.assign($cc.cred, cd$)
17  if $ev = cancel(db, cd, c) \wedge c \in p$ then $p$.remove($c$)
18  if $ev = release(cd, db, c) \wedge c \in p$ then $p$.remove($c$)
19  if $ev = deleg\text{-}no\text{-}resp(db, db_2, c) \wedge c \in p$ then $c.db \leftarrow db_2$
20  if $ev = deleg\text{-}ret\text{-}resp(db, db_2, c) \wedge c \in p$
21     then $p$.addCommI($c.id, db_2, c.cd, c.ant, c.cons, c.args$)
22  if $ev = assign\text{-}no\text{-}cred(cd, cd_2, c) \wedge c \in p$ then $cj.cd \leftarrow cd_2$
23  if $ev = assign\text{-}ret\text{-}cred(cd, cd_2, c) \wedge c \in p$
24     then $p$.addCommI($c.id, c.db, cd_2, c.ant, c.cons, c.args$)
25  for each CommInst $c \in p$
26  do $c$ = ResiduateAntCons($c, ev$)
27      if $c.state$ = fulfilled then enqueue($fulfill(c)$)
28  if $\exists c \in p.finalcomminsts$ s.t. $fulfill(c) \in evts$
29     then for each CommInst $cj \in p$ do $release(cj.db, cj.id)$
30          $p.state \leftarrow$ fulfilled
31          enqueue($fulfill\text{-}p(p.id, p.key)$)
32  if $\forall c \in p.finalcomminsts$ . $c.state$ = violated
33     then for each CommInst $cj \in p$ do $release(cj.db, cj.id)$
34          $p.state \leftarrow$ failed
35          enqueue($failure\text{-}p(p.id, p.key)$)
36  enqueueAll(getImpliedFrom($p.spec, ev$))
```

The antecedent and consequent of commitment instances in the protocol instance are residuated [27]: they are updated based on the fact that a new event has occurred (lines 25–27). If a commitment's consequent is '$p \cdot q$', and event 'p' occurs, the consequent becomes 'q'. If 'q' occurs before 'p', the status of the commitment switches to violated. If a commitment is fulfilled, a corresponding event is enqueued.

Lines 28–36 handle protocol termination. Success (lines 28–31) occurs if a final commitment is fulfilled: active commitment instances are released, the protocol instance is set to fulfilled, and a corresponding event is enqueued. Failure (lines 32–35) occurs if all final commitment instances are violated. Finally, all the events that are implied from the *ev* via implies relationships are enqueued for processing (line 36).

Algorithm 2 Checking compliance with a protocol instance

```
CHECKCOMPLIANCE(Event $ev$, ProtInst $p$)
 1  ProtSpec $sp \leftarrow p.spec$
 2  if $mut\text{-}excl(St_i, Ev) \in sp \ \wedge \ st_i(p.key) \in p.kb$
 3    then $\underline{\text{ERROR}}(p, mut\text{-}excl(st_i, ev))$
 4  for each CommInst $c \in p$
 5  do if NOW $> c.creatDeadline \wedge\ !c.created$
 6       then $\underline{\text{ERROR}}(p, create\text{-}timeout(c))$
 7     if NOW $> c.fulfilDeadline \wedge\ !c.fulfilled$
 8       then $\underline{\text{ERROR}}(p, fulfill\text{-}timeout(c))$
 9     if $ev = deleg\text{-}ret\text{-}resp(db_1, db_2, c) \wedge\ !c.canDelRet$
10       then $\underline{\text{ERROR}}(p, del\text{-}ret(c))$
11     if $ev = deleg\text{-}no\text{-}resp(db_1, db_2, c) \wedge\ !c.canDelNoR$
12       then $\underline{\text{ERROR}}(p, del\text{-}no\text{-}r(c))$
13     if $ev = assign\text{-}ret\text{-}cred(cd_1, cd_2, c) \wedge\ !c.canAssgnR$
14       then $\underline{\text{ERROR}}(p, assign\text{-}ret(c))$
15     if $ev = assign\text{-}no\text{-}cred(cd_1, cd_2, c) \wedge\ !c.canAssgnNoC$
16       then $\underline{\text{ERROR}}(p, assign\text{-}no\text{-}r(c))$
17     if $ev = cancel(db, c) \wedge\ !c.canCancel$ then $\underline{\text{ERROR}}(p, cancel(c))$
18     if $ev = create(db, cd, c)$
19       then CommClass $cc \leftarrow p.$CLASSOF$(c)$
20            if NIL $\neq p.$VALOF$(cc.deb) \neq db$
21              then $\underline{\text{ERROR}}(p, wrong\text{-}deb(c, db))$
22            if NIL $\neq p.$VALOF$(cc.cred) \neq cd$
23              then $\underline{\text{ERROR}}(p, wrong\text{-}cred(c, cd))$
24  for each AgentVar $agv \in p.agent\text{-}vars$
25  do Role $rl \leftarrow agv.role$
26     Agent $ag \leftarrow p.$GETASSIGNEDAGENT$(agv)$
27     if $ag =$ NIL then break
28     if $max\text{-}per\text{-}role(rl, n) \in sp \ \wedge \ |p.$COMMWITHDEB$(ag)| > n$
29       then $\underline{\text{ERROR}}(p, max\text{-}per\text{-}role(ag, rl))$
30     for each CommClass $cc \in sp.comms$
31     do if $max\text{-}of\text{-}class(rl, cc.id, n) \in sp\ \wedge$
32           $|p.$COMMOFTYPEWITHDEB$(cc, ag)| > X$
33          then $\underline{\text{ERROR}}(p, max\text{-}of\text{-}class(ag, rl, cc.id, X))$
34     if $role\text{-}confl(rl, rl_2) \in sp \ \wedge \ p.$PLAYS$(ag, rl_2)$
35       then $\underline{\text{ERROR}}(p, role\text{-}confl(ag, rl, rl_2))$
36     if $comm\text{-}role\text{-}confl(rl, C_1, C_2) \in sp\ \wedge$
37        $p.$PLAYS$(ag, rl_2) \wedge p.$DEBFORBOTH$(ag, C_1, C_2)$
38       then $\underline{\text{ERROR}}(p, comm\text{-}role\text{-}confl(ag, rl, C_1, C_2))$
39  for each $ag \in p.$GETALLPARTICIPANTS$()$
40  do if $sep\text{-}duties(C_1, C_2) \in sp \ \wedge \ p.$DEBFORBOTH$(ag, C_1, C_2)$
41       then $\underline{\text{ERROR}}(p, sep\text{-}duties(ag, C_1, C_2))$
```

Algorithm 2 raises errors whenever an event violates a constraint in the specification of a protocol instance. Lines 2–3 handle mutual exclusion constraints (mut-excl): if the occurred event happens, and the knowledge base entails a conflicting state of affairs, an error is raised. Lines 4–23 examine all commitment instances, and raise errors when different commitment constraints are violated: expired creation deadline ($\leq t$), expired fulfillment deadline (deadline), disallowed delegation with retained responsibility (deleg-ret-resp), disallowed delegation without responsibility (deleg-no-resp), disallowed assignment retaining credit (assign-ret-cred), disallowed assignment without credit retainment (assign-no-cred), and disallowed cancellation (cancel). Lines 18–23 raise errors if the event is the creation of a commitment, but the debtor or the creditor is not the expected one. For instance, if a commitment class has debtor agent variable agv_1, agent "john" is already assigned to agv_1, and a commitment instance for that class is created with debtor "mike", an error is raised. Lines 24–41 detect violations of protocol refinement constraints, such as max-per-role and sep-duties.

V. Implementation

As a proof of concept, we implemented two prototypes that enable creating Azzurra textual and diagrammatic specifications (Section V-A), and the compliance checking algorithms that are used at runtime (Section V-B).

A. Modeling Tool

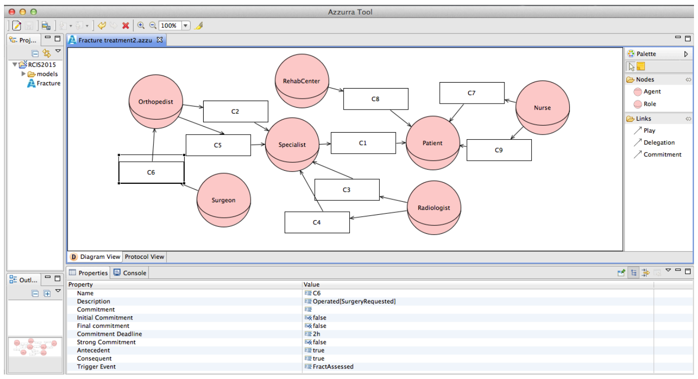
The Azzurra modeling tool is a standalone Eclipse application, built on top of the GEF (Graphical Editing Framework) and XText frameworks. The environment supports the modeling of business processes in terms of *views* that allows the modeler to focus on different aspects of the domain and, thus enables a better separation and representation of concerns during modeling time.

The *social view* (depicted in Fig. 2a) provides an intuitive interface for the modeler, by enabling designers to graphically represent the social relations among the several roles and agents, in terms of their commitments and commitment delegations. Using the Properties tab (below the graphical representation), it is possible to specify commitment's name, antecendent, consequent, triggering event and deadline. Further, one can also specify whether a given commitment is an initial, final or strong commitment.

The *protocol view* (textual view) also enables designers to enrich the specification by capturing other details like triggering events for commitments as well as commitment refinements and parts of the knowledge base of the protocol (depicted in Fig. 2b). Commitment refinements can be captured either by editing the Properties tab or by editing the textual protocol representation as depicted in Fig. 2b. Finally, the tool also enhances the modeling process by enabling the checking of well-formed Azzurra models, detecting invalid commitments and commitment delegations at modeling-time.

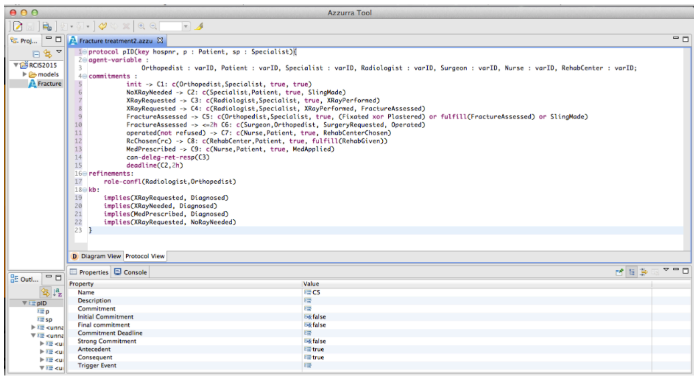
B. Algorithms Implementation and Performance Evaluation

The compliance checking algorithms have been implemented in a prototype Java tool that uses the Drools rule engine to draw inferences and determine which are the active commitments, and whether the agents are acting in compliance with the specification. The tool takes as input a trace of events, and processes it using Algorithm 1 and Algorithm 2 to interpret exchanged messages in terms of protocols and commitments. Whenever an event is read and processed, the tool updates the internal data structures that are stored as Json objects in a MongoDB database. The tool supports multiple protocol classes; in order to add one protocol to the database, a text file is creating following the syntax in Table II, which can



Property	Value
Name	C6
Description	Operated[SurgeryRequested]
Commitment	
Initial Commitment	false
Final commitment	false
Commitment Deadline	2h
Strong Commitment	false
Antecedent	true
Consequent	true
Trigger Event	FractAssessed

(a) *Social view* (graphical representation)



```
protocol pID(key hospnr, p : Patient, sp : Specialist){
agent-variable :
        Orthopedist : varID, Patient : varID, Specialist : varID, Radiologist : varID, Surgeon : varID, Nurse : varID, RehabCenter : varID;
commitments :
      init -> C1: c(Orthopedist,Specialist, true, true)
      NoXRayNeeded -> C2: c(Specialist,Patient, true, SlingMade)
      XRayRequested -> C3: c(Radiologist,Specialist, true, XRayPerformed)
      XRayRequested -> C4: c(Radiologist,Specialist, XRayPerformed, FractureAssessed)
      FractureAssessed -> C5: c(Orthopedist,Specialist, true, (Fixated xor Plastered) or fulfill(FractureAssessed) or SlingMade)
      FractureAssessed -> <=2h C6: c(Surgeon,Orthopedist, SurgeryRequested, Operated)
      operated(not refused) -> C7: c(Nurse,Patient, true, RehabCenterChosen)
      RcChosen(rc) -> C8: c(RehabCenter,Patient, true, fulfill(RehabGiven))
      MedPrescribed -> C9: c(Nurse,Patient, true, MedApplied)
      can-deleg-ret-resp(C3)
      deadline(C2,2h)
refinements:
   role-confl(Radiologist,Orthopedist)
kb:
   implies(XRayRequested, Diagnosed)
   implies(XRayNeeded, Diagnosed)
   implies(MedPrescribed, Diagnosed)
   implies(XRayRequested, NoRayNeeded)
}
```

Property	Value
Name	C5
Description	
Commitment	
Initial Commitment	false
Final commitment	false
Commitment Deadline	
Strong Commitment	false
Antecedent	true
Consequent	true
Trigger Event	

(b) *Protocol view* (textual representation)

Fig. 2: Views of fracture treatment scenario (Fig. 1) using the Azzurra modeling language

be exported from the modeling environment shown in Fig. 2a and Fig. 2b. The tool uses a parser—generated with Antlr—to populate the database.

We have conducted preliminary tests on the performance of our runtime tool. The results have shown that performance is not affected by the number of events processes so far (hence, by the number of active commitment instances); it takes just a handful of milliseconds (between 5 and 100) to process an event. On the other hand, protocol instantiation events affect significantly the performance of the tool, for they require instantiating the protocol itself, its initial commitments, binding agents to variables, initializing constraints, etc., through the creation of a number of Json objects and their storage in the database. On average, it takes a few seconds (5 to 10) to instantiate a protocol like that in Table II.

The runtime tool and its performance evaluation are too preliminary to draw any conclusions on the scalability of the framework. This is part of our current and future work.

VI. Evaluation on Scenarios

We conducted a preliminary evaluation of Azzurra's applicability by modeling two scenarios that have been extracted from two different real-world cases within the medical domain, followed by a general discussion on Azzurra. The healthcare domain has been selected due to the recognition of one of the most promising, but still challenging domains for the adoption of process-oriented solutions due to complex needs stemming from the business domain [11].

The first scenario (Section VI-A) compares Azzurra's representational features to those of the three main types of process modeling languages: (activity-centered) imperative, (activity-centered) declarative and (artifact-centered) declarative modeling languages, see Section VII for details. Therefore, we have selected the most prominent representative of each category to establish our comparison. Our goal is to demonstrate in which aspects Azzurra conceptualization differs from the other representational methods.

The second scenario (Section VI-B), on the other hand, emphasizes certain domain characteristics of the scenario that could be better supported by a commitment-based representation. For that, we depict imperative process models as a means of presenting these domain characteristics and then, contrast the imperative representation with the corresponding commitment-based representation.

A. Fracture Treatment Scenario

In this first scenario, we present alternative models of the fracture treatment example from the literature [31], and compare them with the Azzurra model presented earlier (Fig. 1).

Fig. 3(a) depicts an operational model for the running example using the BPMN modeling language. Imperative languages represent business processes in terms of activities to be executed as well as the exact sequence between these activities. Here, the model consists of activities (e.g., "Examine patient" and "Verify need of medication") and the control flow among them. Since activities must be explicitly activated for enactment, this type of representation requires an explicit (… and exhaustive) specification of all possible enactment paths. For instance, a recurrent enactment path for our example is "Examine patient" and then "Verify need of medication", but there are many others as well (not represented in Fig. 3(a)). Azzurra models enable more flexible specifications of process models because it only requires the specification of essential ordering constraints between commitments. For instance, in Table II and Fig. 1, only C_1, C_4 and C_6 include temporal constraints. Further, as commitments can be satisfied by different activities, the "Examine patient" commitment could be fulfilled through different operationalizations as for instance, the doctor could first "Perform a physical evaluation" and subsequently "Examine patient's family history" or alternatively, s/he could perform the same activities in the inverse order.

Unlike imperative languages, declarative ones require only the minimal set of constraints between activities. By default, all execution paths are allowed and prohibited execution paths are specified by constraints on the execution order between activities. Fig. 3(b) (extracted from [31]) presents the declarative specification of our running example using DECLARE. Azzurra is also declarative like DECLARE, but it does not focus on activities for expressing business processes, rather emphasizing their social nature by capturing agents

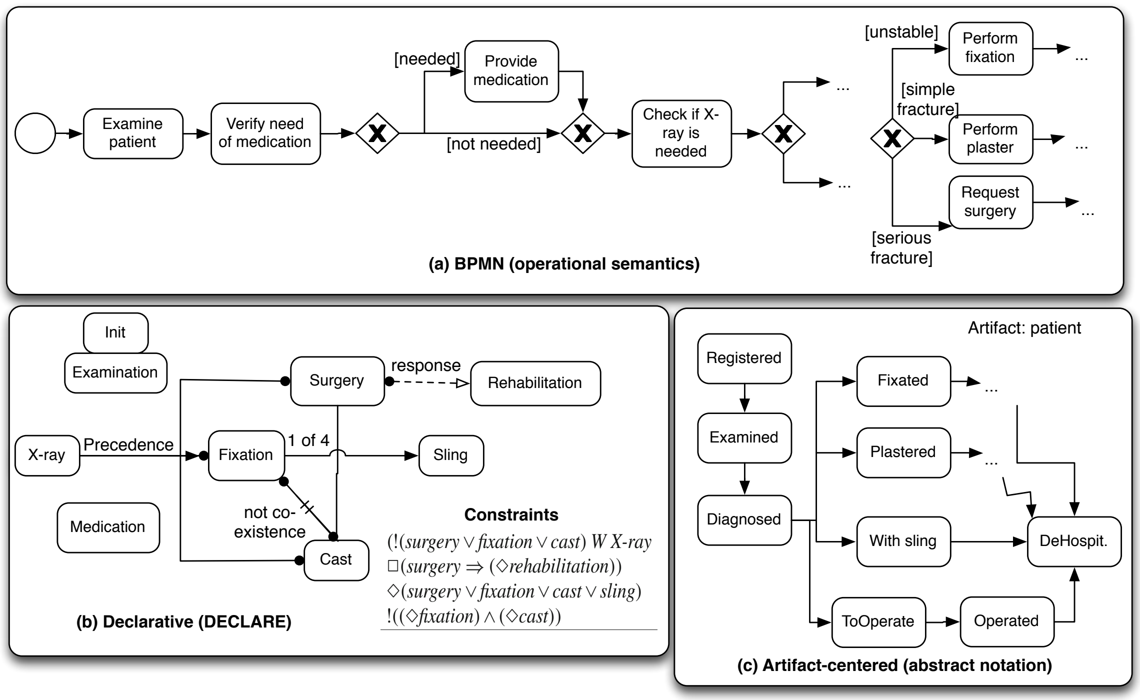


Fig. 3: Snippets of the fracture treatment process using (a) a operational workflow language; (b) a declarative language; (c) an artifact-centered notation

and commitments between them. The approach of modeling business processes in terms of commitments among process participants also increases flexibility in the specification as in the imperative paradigm, once it does not constrain process participants to execute particular activities during runtime, but instead, it expands the number of operational choices as long as these activities satisfies the commitments among agents.

Differently from its activity-centered cousins, the artifact-centered paradigm promotes data objects to first-class citizens in modeling a process, by describing the lifecycle of each object. Here, activities that change/update the state of an object are also represented. In our example, fracture treatment is represented as a data object called “Patient” with several interconnected states. The control flow of the business process does not have to be exhaustively modelled, relying instead on the lifecycle model of the data objects: “registered”, “examined”, . . . , “dehospitalized” (see Fig. 3(c)). The states of the data object are similar to the propositions in the Azzurra version of the process (e.g., “examined”, “diagnosed” in Table II). However, by centering the representation on artifacts, the business process has an operational perspective. Differently, Azzurra’s commitment-based representation highlights the social nature of business processes, representing who is responsible for advancing the state (the debtor in a commitment). Further, while the artifact-centered paradigm focuses on the activities that change the states of data objects, Azzurra focuses on correctness criteria rather than specific operationalizations. This approach favors flexibility as different activities are admissible at runtime, as long as they satisfy the correctness criteria stipulated by the commitments.

B. *Clinical Guidelines Scenario*

In our second scenario, we also consider a business process from the medical domain that concerns Clinical Guidelines (CGs). CGs consists of “systematically developed statements to assist practitioner and patient decisions about appropriate health care for specific clinical circumstances” [15]. In the context of a CG, every activity in the process model corresponds to a recommendation that support healthcare providers (doctors, nurses, etc.) to develop care actions for patients. Therefore, every activity within the process model can be understood as an abstract recommendation (abstract activity) to be adapted at runtime according to a specific patient by the healthcare provider executing the CG. Given the abstract nature of CGs that require extensive adaptation of abstract activities at runtime, we say that CGs are inherently *decision-intensive* business processes.

Figure 4(a) depicts an example of an executable clinical guideline for transient ischemic attack (TIA) (an episode of neurological occurrence) from the literature [33], [36]. The CG is represented in terms of activities for each recommendation and executing constraints between these activities using the BPMN notation. To exemplify the decision-intensive nature of a CG, consider the “Treat for stroke” recommendation/activity. During process execution, this recommendation has to be personalized for a specific patient, considering (i) the execution context (ii) doctor’s expertise and (iii) patient’s clinical circumstances. For instance, assuming that there are two procedures for treating stroke (“surgery” and “endovascular procedure”), the doctor has to select the best alternative for the patient by considering environmental constraints, such as the availability

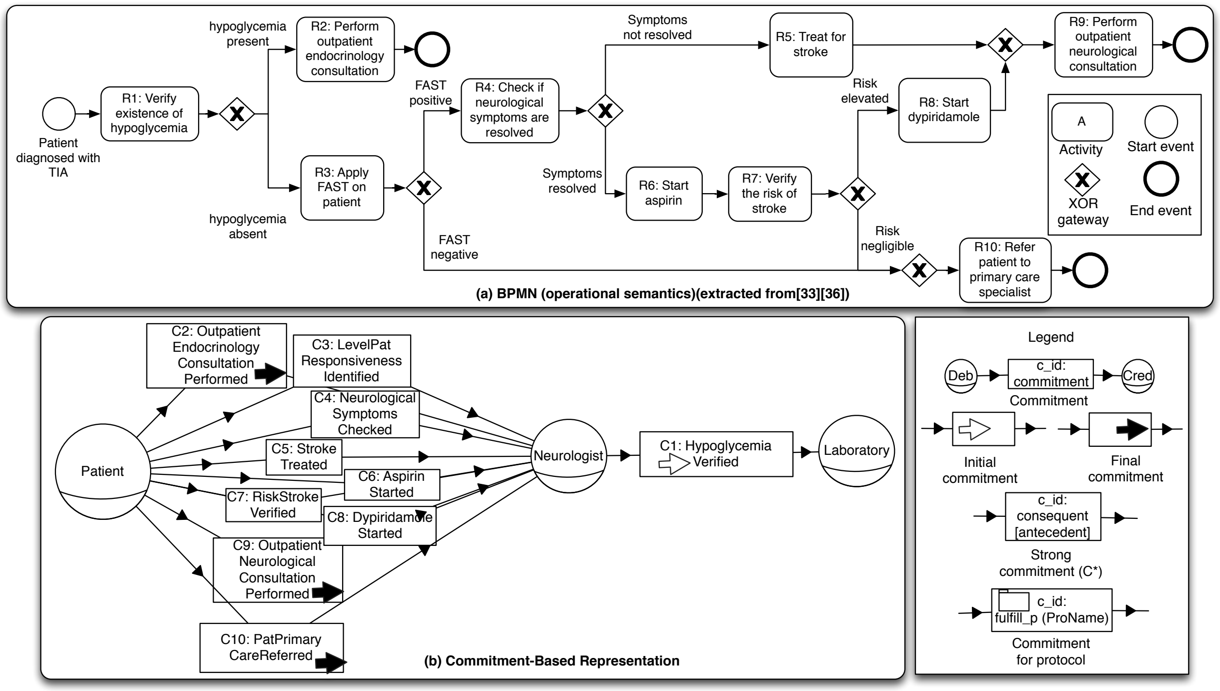

Patient diagnosed with TIA
R1: Verify existence of hypoglycemia
hypoglycemia present
hypoglycemia absent
R2: Perform outpatient endocrinology consultation
R3: Apply FAST on patient
FAST positive
FAST negative
R4: Check if neurological symptoms are resolved
Symptoms not resolved
Symptoms resolved
R5: Treat for stroke
R6: Start aspirin
R7: Verify the risk of stroke
Risk elevated
Risk negligible
R8: Start dypiridamole
R9: Perform outpatient neurological consultation
R10: Refer patient to primary care specialist
A
Activity
Start event
XOR gateway
End event
(a) BPMN (operational semantics)(extracted from[33][36])
C2: Outpatient Endocrinology Consultation Performed
C3: LevelPat Responsiveness Identified
C4: Neurological Symptoms Checked
C5: Stroke Treated
C6: Aspirin Started
C7: RiskStroke Verified
C8: Dypiridamole Started
C9: Outpatient Neurological Consultation Performed
C10: PatPrimary CareReferred
Patient
Neurologist
C1: Hypoglycemia Verified
Laboratory
(b) Commitment-Based Representation
Legend
Deb
c_id: commitment
Cred
Commitment
Initial commitment
Final commitment
c_id: consequent [antecedent]
Strong commitment (C*)
c_id: fulfill_p (ProName)
Commitment for protocol

Fig. 4: Transient Ischemic Attack (TIA) Clinical Guideline using (a) a operational workflow language (BPMN) and (b) a commitment-based representation

of procedures and/or costs of each of them.

Most of the languages for representing CGs follow a task-based paradigm in which recommendations are represented as actions and decisions in a rigid flowchart-like (imperative) structure [11], like the BPMN representation in Figure 4(a). However, adopting this approach indeed introduces a number of shortcomings in the CG representation from a domain point of view, that are required by imperative process languages, like inexistent ordering constraints between multiple recommendations. Although it is out of scope of this paper to provide a more extensive discussion about the topic of CG representation, our intention here is to demonstrate how a commitment-based approach could help to tackle some of the problems with the imperative representation. For that, we introduce in Figure 4(b) the respective commitment-based representation of Figure 4(a). While activities in CGs represent recommendations for healthcare providers on how to address particular clinical circumstances, commitments instead capture these recommendations as a compromises of the healthcare provider who is executing the guideline towards the patient (and also the compromises of other healthcare providers in the scope of the guideline).

In the remainder, we point out some of the shortcomings introduced by imperative languages and contrast the corresponding representation with the Azzurra model:

- *Negative recommendations:* Imperative process models (Figure 4(a)) describe recommendations like “Treat for stroke” and “Apply FAST on patient” as activities. This approach works well for positive recommendations, i.e., actions that have to be performed. Differently, for negative recommendations as “Do not provide aspirin” (which is admissible from a business perspective [36]), the activity-based representation fails. Indeed, the existence of negative recommendations suggest that recommendations are not actions themselves, but rather positive and negative restrictions on the behavior (actions). By centering the representation on commitments, Azzurra specifies restrictions on behavior, defining correctness criteria that should not be violated. In this case, the issue with negative recommendations can be solved by specifying a commitment whose consequent in a negative correctness criteria (for example, $\neg AspirinProvided$);
- *Ordering constraints:* As the imperative representation represents recommendations as actions that have to be performed (and actions are represented in sequence within the imperative paradigm), the paradigm imposes a natural sequence among these recommendations. From a domain perspective, however, ordering constraints among recommendations are not necessary or even desirable [36] (this lack of sequence can be indeed evidenced by the existence of negative recommendations). Differently, Azzurra does not impose any order among commitments, but when necessary, they can be specified by matching commitments consequent and antecedent;
- *Conflicting recommendations:* in the imperative representation, recommendations are modeled as labeled activities (textual information) and no mechanisms are specified to correlate related actions (for instance, “Provide aspirin” and “Do not provide aspirin” are modeled as unrelated actions in the specification). As a consequence of that, external rules must be defined to capture conflicting actions, whereas automatic detection could be performed

by reasoning over the meaning of the actions [36]. In a commitment-based approach, as commitment's consequent capture recommendations (for instance, for a recommendation "Provide aspirin", the commitment consequent is $AspirinProvided$), conflicting recommendations could be automatically detected. For instance, in a hypotethical situation in which aspirin conflicts with clopidogrel, the knowledge base could capture this conflict as a rule and design-time model-checking techniques could be applied to reason about conflicting commitments (for example, two commitments whose consequent are $AspirinProvided$ and $ClopidogrelProvided$ cannot exist in the same Azzurra specification). Alternatively, other conflicting recommendations could also be detected, like "Provide aspirin" ($AspirinProvided$) and "Do not provide aspirin" ($\neg AspirinProvided$).

- *Compliance checking:* As a guideline specification is intended to provide recommendations for healthcare providers to execute actions, from a practical point of view, they have freedom to either change the suggested care actions (i.e., change the actions that satisfy a given recommendation/commitment) or even to completely skip certain recommendations when necessary. However, compliance to guidelines is assessed in a strict manner by only matching recommended actions with executed actions [29]. This means that, although they are free to select the best care actions at runtime, substitutions in the recommendations will accuse false cases of non-compliance. Azzurra leverages compliance to the business level, by not specifying concrete actions to be executed, but rather correctness criteria. This opens the possibility of using alternative actions to fulfill the commitment (depending how suitable they are in relation to the executing context), as long as they satisfy the commitments. Furthermore, by capturing actors and their commitments, accountability can be easily checked in an Azzurra specification. This is also fundamental in a medical context, once responsibility for care actions need to be strictly tracked along the treatment process.

C. Discussion

The fracture treatment scenario shows how Azzura natively supports modeling business processes in the healthcare domain; this style of modeling has advantages in other domains too. Unlike current languages, that center their representation either in *activities* or *data objects*, Azzurra captures the *social nature* of the interactions between process participants by expressing these interactions in terms of commitments (correctness criteria based on social expectations). Centering the representation in terms of activities/data objects leads to an operational business process representation, once the behavior is specified in terms of specific operationalizations to achieve the desired outcomes, rather than *what* is supposed to be achieved. As a general consequence of the shift in the representation, specifications in Azzurra allow one to capture business processes in more strategic terms. In particular, the benefits of such approach in the first scenario can be manifested as (i) the ability of focusing on the social perspective of the business processes, (ii) it enables a more flexible representation of the process than its respective counterparts in other process languages. This flexibility is manifested through the ability of specifying different sequences of commitments that can be satisfied by different concrete activities.

While in the first scenario Azzurra provides increased flexibility for business process specification, the second scenario demonstrates that a shift in the modeling paradigm is rather fundamental to address the representational needs (knowledge structure) of clinical guidelines. To enumerate the CG flexibility needs more concretely, guidelines are inherently *decision-intensive* and act as abstract templates/blueprints that provide evidence-based decision support for healthcare providers. They do not prescribe the actual behavior within the business process, but rather *constraints on the behavior* and require subsequent adaptation and personalization to obtain a concrete medical treatment (actions) for a given patient [11]. As a result, it is not possible to define a priori all the variants in the execution of a business process (imperative modeling would require doing so). Azzurra, on the other hand, represents these guidelines through correctness criteria in terms of commitments.

In summary, Azzurra better supports not only the representation of the knowledge structure of the domain (by being able of representing negative recommendations as well as the essential ordering constraints), but also presents an advantage for the reasoning techniques that must be executed on the basis of CG models, as reasoning about conflicting recommendations and checking compliance. More specifically, considering compliance checking, Azzurra expands the notion of compliance to the business level, once correctness criteria allows one to consider different actions that satisfy commitments and not to necessarily stick to one particular activity as it is done in the current practice. Moreover, relying on commitments between agents, Azzurra natively supports accountability, i.e., enables determining at all times which agents are compliant, and which ones have violated a commitment they are responsible for.

VII. Related work

Proposals targeting the representation of distributed behavior among agents are mainly stemmed from the Business Process Management (BPM) area within the umbrella of business process modeling languages. Historically, Petri Nets were probably the first formalism to treat concurrency as first-class citizen [32]. Despite the availability of well-established formalisms like Petri Nets and process calculi, industry needs pushed the adoption of a plethora of conceptual languages like BPMN, BPEL, UML, EPCs, workflow nets, etc [32]. The majority of conceptual process modeling languages model business processes within an imperative paradigm which is basically founded on the notions of activities and control flows among them. Such languages are intended to define operational details to better support process execution and work well for structured and repetitive business processes.

The rigidity imposed by the imperative paradigm that requires exhaustive specification of all possible paths during design-time triggered efforts for the development of declarative languages. In this context, declarative workflows [31] have arisen as a alternative for providing more flexible specifications of business processes, by enabling the representation of behavior in terms of minimal precedence constraints among activities.

The realization that the primary driver for the progress of certain types of business processes is not the event related to the completion of activities, but instead the availability of certain values of data objects [22] led to the creation of the artifact-centered paradigm. This paradigm proposes a hybrid approach for the representation of business processes, by capturing them in terms of activities and artifacts (data objects). More specifically, Bhattacharya et al. [4] make artifacts such as purchase orders, invoices, etc., as the focus of business process modeling, and define workflows that represent the lifecycle of these artifacts.

Process modeling languages as well as novel paradigms such as declarative and artifact-centered approaches originate from technical requirements concerning process execution (e.g., the introduction of the declarative paradigm motivated by the issue of ordering constraints imposed by the imperative paradigm). Differently, Azzurra is motivated by need of providing a social perspective of business process representation, by capturing these business processes in terms of intentional agents and the expectations of these agents towards each other, i.e., their commitments. As a result, instead of expressing how to achieve a determined business goal through a prescription of a number of steps (activities), Azzurra specifies the constraints that have to be respected and gives the participating agents the autonomy to decide the best operationalizations to achieve the outcomes during runtime. This shift in the modeling paradigm opens up the possibility of providing more flexible specifications for business processes as demonstrated in Section VI-C.

Still in the area of BPM, a number of approaches are also related to Azzurra in different ways. First, as Azzurra represents autonomous agents and their interactions through commitments and protocols, choreographies are also a related approach as it specifies the flow of messages among autonomous actors. Van der Aalst [30] advocates choreographies for modeling cross-organizational business processes. Khalaf [18] shows how to map the RosettaNet PIPs business protocols to abstract BPEL processes. Decker et al. [8] extend BPEL with choreography-related constructs. WS-CDL [34] and BPMN 2.0 both support the specification of choreographies. Benatallah et al. [3] propose a transition-based conversation model to conceptualize web service conversations. Unlike choreographies, Azzurra captures the meaning of interaction in terms of commitments among actors.

As commitments are triggered by events in Azzurra, event-driven business chains [17] also constitute an alternative to activity-based processes, by centering on events that trigger functions performed by organizational units. Our approach considers high-level events that update the commitments of actors.

By centering on autonomous actors that interact to perform a business process, social and resource management perspectives in BPM also present some similarities with Azzurra. Cabanillas et al. [6] introduces the Resource Assignment Language (RAL). The RAL language (which has a formal semantics based on Description Logics) extends the BPMN language with RAL expressions in order to provide mechanisms for history-based human resources management within business processes. Similarly, Brambilla et al. [5] also extend the BPMN notation for capturing social requirements, by including new events and task types together with some annotations for pools and lanes. Differently from both proposals that extend BPMN notation with human resource constructs, Azzurra indeed proposes the representation of business processes in terms of commitments among actors.

As commitments carry a contractual meaning from the social point of view, compliance management approaches in BPM have also been investigated. In that respect, Ghose and Koliadis [14] annotate business processes with constraints on their execution in order to define normative compliance. Sadiq et al. [25] enrich business process models with obligations that members of an enterprise must fulfill in order to remain compliant. While a commitment is a social relation between actors, these obligations are technical constraints on information system design.

In other areas the topic of commitments have also been largely explored. In cooperative work, commitments constitute the main abstraction in the influential *Coordinator* system [13], intended to track activities within an organization. This work inspires our approach. However, the Coordinator adopts an operational view of conversations, which is less flexible than the commitment protocols proposed here.

Commitments are also an important abstraction in the design of multi-agent systems for specifying and analyzing the interactions between several autonomous agents. As Azzurra, Desai et al. [10] and Yolum [35] use commitments and protocols as design abstractions for business processes. These works inspire the REGULA framework [19], which introduces temporal operators to represent more expressive commitments and reasoning about them. A number of works also use commitments for specifying cross-organizational business processes. In [9], Desai at al. describe the Amoeba methodology for specifying business processes based on business protocols. Robinson and Purao [23] propose a framework for specifying and monitoring cross-organizational business processes that relies upon commitments enriched with a temporal logic. Nandi and Sanz [20] employ sets of commitments as cross-organizational contracts that lead to value creation. Azzurra's novelty compared to this work includes: (i) advanced primitives for expressing business patterns such as separation of duties, compensations, workload limits; also lifetime support for protocol instances, from initiation to termination; (ii) a graphical notation to visualize the main elements of a protocol; and (iii) algorithms for determining compliance of observed behavior with a specification.

VIII. Conclusion

We have presented Azzurra, a specification language for business processes founded on the concept of social commitment. Azzurra comprises business primitives that focus on the obligations process participants have towards each other, rather than on activities to be carried out, as well as delegations, deadlines, and role adoption constraints. In addition to syntax and semantics for Azzurra, we have proposed a graphical notation to visualize the main elements of an Azzurra specification, and algorithms for checking whether the interactions among a set of actors comply with a protocol specification.

This paper opens the door to further work on social models for business processes. We plan to continue research on Azzurra with work that (i) develops an enactment engine that supports remedies to noncompliance; (ii) conducts an empirical evaluation of Azzurra with industrial case studies; (iii) improves the graphical notation; (iv) investigates the joint usage of Azzurra specifications and operational business process models (written, for example, in BPMN) and with business artifacts; and (v) analyze the derivation of Azzurra protocols from organizational goals.

ACKNOWLEDGMENT

This work has been partially supported by the ERC advanced grant 267856 for a project titled "Lucretius: Foundations for Software Evolution", April 2011–March 2016 (http://www.lucretius.eu). We also would like to thank Amit Chopra for contributions in the early stages of the research here reported, Dagimawi Neway for the implementation of the Azzurra Modeling tool and Veruska Zamborlini for proofreading the section about clinical guidelines.

REFERENCES

[1] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana, "Business Process Execution Language for Web Services," IBM, Tech. Rep., 2003. [Online]. Available: http://www.ibm.com/developerworks/library/ws-bpel

[2] H. S. Becker, "Notes on the Concept of Commitment," *American Journal of Sociology*, vol. 66, pp. 32–40, 1960.

[3] B. Benatallah, F. Casati, F. Toumani, and R. Hamadi, "Conceptual Modeling of Web Service Conversations," in *Proc. of CAiSE*, 2003, pp. 449–467.

[4] K. Bhattacharya, C. Gerede, R. Hull, R. Liu, and J. Su, "Towards Formal Analysis of Artifact-Centric Business Process Models," in *Proc. of BPM*, 2007, pp. 288–304.

[5] M. Brambilla, P. Fraternali, and C. Vaca, "A Notation for Supporting Social Business Process Modeling," in *Proc. of BPMN workshop*, 2011, pp. 88–102.

[6] C. Cabanillas, M. Resinas, and A. Ruiz-Cortés, "Designing Business Processes with History-Aware Resource Assignments," in *Proc. of BPM Workshops (BPD'12)*, 2012, pp. 101–112.

[7] A. K. Chopra, F. Dalpiaz, P. Giorgini, and J. Mylopoulos, "Modeling and Reasoning about Service-Oriented Applications via Goals and Commitments," in *Proc. of CAiSE*, 2010, pp. 113–128.

[8] G. Decker, O. Kopp, F. Leymann, and M. Weske, "Interacting Services: From Specification to Execution," *Data and Knowledge Engineering*, vol. 68, pp. 946–972, 2009.

[9] N. Desai, A. K. Chopra, and M. P. Singh, "Amoeba: A Methodology for Modeling and Evolution of Cross-Organizational Business Processes," *ACM Transactions on Software Engineering and Methodology*, vol. 19, no. 2, 2009.

[10] N. Desai, A. U. Mallya, A. K. Chopra, and M. P. Singh, "Interaction Protocols as Design Abstractions for Business Processes," *IEEE Transactions on Software Engineering*, vol. 31, pp. 1015–1027, 2005.

[11] C. Di Ciccio, A. Marrella, and A. Russo, "Knowledge-Intensive Processes: An Overview of Contemporary Approaches," in *Proc. of KiBP workshop*, 2012, pp. 33–47.

[12] K. Figl and R. Laue, "Cognitive Complexity in Business Process Modeling," in *Proc. of CAiSE*, 2011, pp. 452–466.

[13] F. Flores, M. Graves, B. Hartfield, and T. Winograd, "Computer Systems and the Design of Organizational Interaction," *ACM Transactions on Information Systems*, vol. 6, pp. 153–172, 1988.

[14] A. K. Ghose and G. Koliadis, "Auditing Business Process Compliance," in *Proc. of ICSOC*, 2007, pp. 169–180.

[15] T. Greenhalgh, *How to Read a Paper: The Basics of Evidence-based Medicine*. John Wiley & Sons, 2014.

[16] C. B. Jones, *Systematic Software Development using VDM*. Prentice Hall Englewood Cliffs, 1990.

[17] G. Keller, M. Nüttgens, and A.-W. Scheer, "Semantische Prozessmodellierung auf der Grundlage "Ereignisgesteuerter Prozessketten (EPK)"," *Veröffentlichungen des Instituts für Wirtschaftsinformatik*, vol. 89, 1992.

[18] R. Khalaf, "From RosettaNet PIPs to BPEL Processes: A Three Level Approach for Business Protocols," *Data and Knowledge Engineering*, vol. 61, pp. 23–38, 2007.

[19] E. Marengo, M. Baldoni, C. Baroglio, A. K. Chopra, V. Patti, and M. P. Singh, "Commitments with Regulations: Reasoning about Safety and Control in REGULA," in *Proc. of AAMAS*, 2011, pp. 467–474.

[20] P. Nandi and J. Sanz, "Cross-Functional Operations Modeling as a Nexus of Commitments: A New Approach for Improving Business Performance and Value-creation," in *Proc. of CBI*, 2013, pp. 234–241.

[21] Object Management Group, *Business Process Model and Notation (BPMN) 2.0*, http://www.omg.org/spec/BPMN/2.0, Object Management Group Std., 2011.

[22] M. Reichert and B. Weber, *Enabling Flexibility in Process-Aware Information Systems - Challenges, Methods, Technologies*. Springer, 2012.

[23] W. N. Robinson and S. Purao, "Specifying and Monitoring Interactions and Commitments in Open Business Processes," *IEEE Software*, vol. 26, pp. 72–79, 2009.

[24] M. L. Rosa, P. Wohed, J. Mendling, A. H. M. ter Hofstede, H. A. Reijers, , and W. M. P. van der Aalst, "Managing Process Model Complexity Via Abstract Syntax Modifications," *IEEE Transactions on Industrial Informatics*, vol. 7, pp. 614–629, 2011.

[25] S. Sadiq, G. Governatori, and K. Namiri, "Modeling Control Objectives for Business Process Compliance," in *Proc. of BPM*, 2007, pp. 149–164.

[26] M. P. Singh, "An Ontology for Commitments in Multiagent Systems: Toward a Unification of Normative Concepts," *Artificial Intelligence and Law*, vol. 7, pp. 97–113, 1999.

[27] ——, "Distributed Enactment of Multiagent Workflows: Temporal Logic for Web Service Composition," in *Proc. of AAMAS*, 2003, pp. 907–914.

[28] J. M. Spivey, "An Introduction to Z and Formal Specifications," *Software Engineering Journal*, vol. 4, pp. 40–50, 1989.

[29] J. van de Klundert, P. Gorissen, and S. Zeemering, "Measuring Clinical Pathway Adherence," *Journal of Biomedical Informatics*, vol. 43, pp. 861–872, 2010.

[30] W. M. P. van der Aalst, M. Dumas, A. H. M. ter Hofstede, N. Russell, H. M. W. Verbeek, and P. Wohed, "Life After BPEL?" in *Proc. of WS-FM workshop*, 2005, pp. 35–50.

[31] W. M. P. van der Aalst, M. Pesic, and H. Schonenberg, "Declarative Workflows: Balancing between Flexibility and Support," *Computer Science-Research and Development*, vol. 23, pp. 99–113, 2009.

[32] W. M. P. van der Aalst, "Business Process Management: A Comprehensive Survey," *ISRN Software Engineering*, vol. 2013, p. 37 pages, 2013.

[33] S. Wilk, M. Michalowski, W. Michalowski, M. M. Hing, and K. Farion, "Reconciling Pairs of Concurrently Used Clinical Practice Guidelines Using Constraint Logic Programming," in *Proc. of AMIA*, 2011, pp. 944–953.

[34] WS-CDL, *Web Services Choreography Description Language Version 1.0*, www.w3.org/TR/ws-cdl-10/, W3C Std., 2005.

[35] P. Yolum and M. P. Singh, "Flexible Protocol Specification and Execution: Applying Event Calculus Planning using Commitments," in *Proc. of AAMAS*, 2002, pp. 527–534.

[36] V. Zamborlini, M. Da Silveira, C. Pruski, A. ten Teije, and F. van Harmelen, "Towards a Conceptual Model for Enhancing Reasoning About Clinical Guidelines: A Case-Study on Comorbidity," in *Proc. of KR4HC workshop*, 2014, pp. 29–44.