UNIVERSITY
OF TRENTO

DIPARTIMENTO DI INGEGNERIA E SCIENZA DELL'INFORMAZIONE

38123 Povo – Trento (Italy), Via Sommarive 5
http://www.disi.unitn.it

Modeling Business Processes with Azzurra: Order Fulfilment

Giulia Canobbio and Fabiano Dalpiaz

December 2012

Technical Report # DISI-12-040

Modeling Business Processes with Azzurra: Order Fulfilment

Giulia Canobbio[1] and Fabiano Dalpiaz[1,2]

[1]DISI, University of Trento, Italy
[2]Department of Computer Science, University of Toronto, Canada

December 5, 2012

Contents

1 Introduction

Azzurra is 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 models are social models consisting of networks of commitments. As such, Azzurra models support the flexible enactment of business processes, and provide a semantic notion of actor accountability and business process compliance. The syntax of Azzurra is shown in Appendix A.
In this technical report, we apply Azzurra to the order fulfilment exemplar [3] that has been introduced in the context of the influential YAWL project [6].

2 Order fulfilment

The business process about *Ordering Fulfilment* includes multiple sub-processes: Ordering, Carrier, Freight in Transit, Payment and Freight Delivered. Figure 1 shows the overall process in YAWL, taken from the reference document. In this technical report, we present an Azzurra specification for the first two sub-processes, i.e., *Ordering* and *Carrier*.

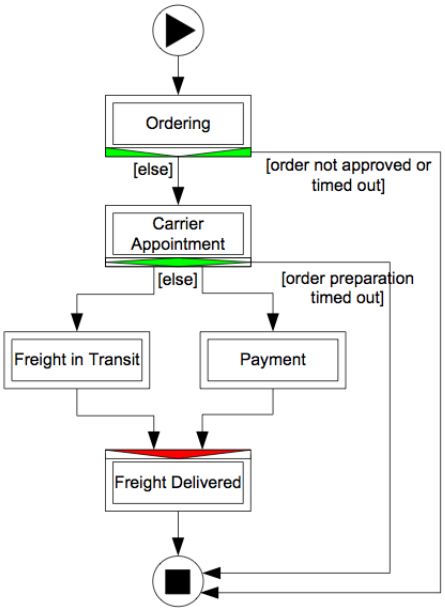

Figure 1: Overall process in YAWL

2.1 Ordering

Ordering is the first business process which is initiated as soon as the protocol *Order Fulfilment* starts. Below, we report the description from [3]:
"The Ordering sub-process starts with the creation of a Purchase Order by the Order Management Department. A Purchase Order (PO) needs to be approved by the Supply Department and may then be subject to a number of modifications, though it requires confirmation within a certain time frame. The creation of a PO is handled by an OD clerk, who may choose to reallocate the task to another PO Manager with or without the work performed on it thus far. A PO Manager may also choose to relinquish

working on the task and have the system offer it again to the available PO Managers. Moreover, a PO Manager may suspend working on the creation of a PO and choose to resume working on it at some later stage. Finally, a PO Manager may volunteer to be the main entry point for processing POs during a certain period of time. When a PO Manager is offered a task to create a PO, and they volunteer for it, the system will initiate the task automatically. Upon completing the PO, the PO Manager needs to decide which PO Manager will work on modification requests as they may eventuate at a later stage. The default PO Manager for PO modifications and confirmations is Carmine Marino (user id cm). The completed PO is passed on to a Supply Officer who needs to approve it. If the Supply Officer who allocates this task to themselves is the Head of SD, they may choose to delegate this task to an SD clerk who reports to them. Supply Officers choose which approval tasks they will work on and once they have chosen such a task they may decide when to actually start work on it. This interaction pattern with the system is the default one for the various tasks that need to be performed in the Genko Oil company. A PO contains information about the client's company (e.g. name, address and business number), information about the order (e.g. order number and date, currency, order terms, line items), the freight cost and the delivery location. Moreover, it is possible to specify whether the order needs to be invoiced and whether it is part of a prepayment agreement between the client and Genko Oil. Once a PO has been approved, repeated modifications may be requested. These are tracked by a revision number attached to the PO which is increased at each change. Each of these changes again need to be approved. If the original PO or any modification is rejected, the order process ends. Moreover, the PO needs to be confirmed within 3 days otherwise it is discarded and the process terminates."

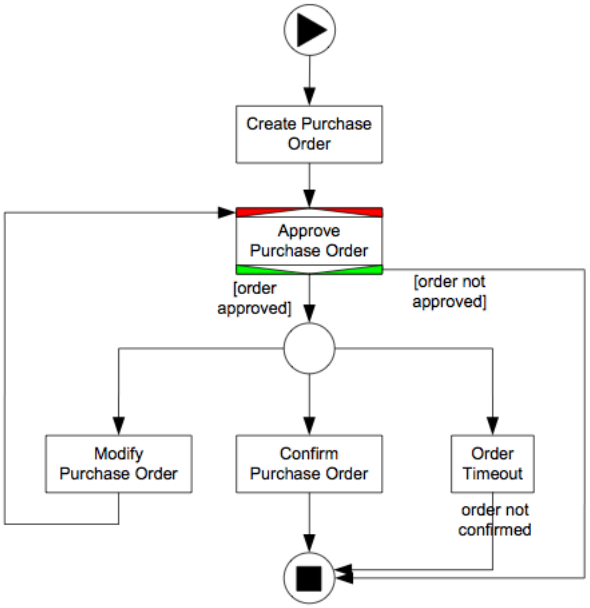

Figure 2: Ordering process in YAWL

Figure 2 shows the ordering business process as a workflow in YAWL; such specification details *how* the process is to be conducted. Azzurra takes a different standpoint, and it specifies the business processes as a set of commitments [4] among participating actors. As such, Azzurra specifies *what* should be done and *who* is accountable for carrying out the *what*. As long as the commitments are fulfilled, participants are free to define their internal workflow.
Table 1 shows the ordering process modeled as an Azzurra protocol, while Figure 3 illustrates the graphical notation of Azzurra. The context of the protocol is genko, i.e., the oil company. The agent variables

protocol Ordering (**context** genko, supplyOff : SupplyOfficer, orDepClerk: OrderDepartmentClerk) {
ag-variables: pom : PurchaseOrderManager;
commitments:
init $\twoheadrightarrow$ C_1:C(orDepClerk, supplyOff, $\top$, POCreated(1))
POUpdated(revNr) $\twoheadrightarrow$ C_2:C(supplyOff, pom, $\top$, POExamined(revNr) $\cdot$ (POApproved(revNr) $\vee$ PORejected(revNr)))
POExamined(revNr) $\twoheadrightarrow$ C_3:C*(pom, supplyOff, POApproved(revNr), POModified(newRevNr) $\vee$ POConfirmed(revNr))
deadline(C_2, 3 days)
kb:
implies(POCreated, POUpdated)
implies(POModified, POUpdated)
mut-excl(POApproved, PORejected)
}

Table 1: Azzurra specification for the ordering process

in the parameters are a supply officer and an order department clerk. In other words, the protocol instantiation requires those two agent variables to be bounded to an actual agent. There is an additional unbounded variable, i.e., the purchase order manager. As soon as the protocol is initiated, the order department clerk bounded to orDepClerk commits (C_1)to the supply officer that a purchase order is created. C_2 says that the supply officer commits to a purchase order to examined an order (either when it is created or after following modifications), and then approved or rejected it. Notice that C_2 binds a purchase order manager to agent variable pom. That purchase order manager C_3 is created by the purchase order manager the purchase order is approved, then it can be modified or confirmed. Moreover, C_2 has a deadline of three days.
Notice how, unlike workflows, Azzurra focuses on the social relationships (commitments) that need be established and fulfilled among actors in the organization, as opposed to saying which is the sequence of tasks to be carried out. The graphical notation shows agent variables in the protocol (actor stick figure), commitments (rectangles with a contract icon, showing the consequent of the commitment)

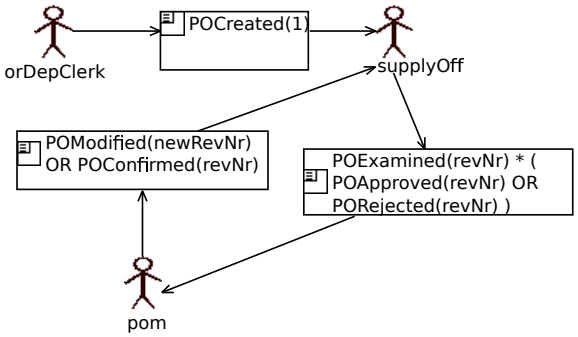

Figure 3: Ordering process in Azzurra (graphical representation)

2.2 Carrier appointment

Carrier appointment is the second business process that is initiated upon the correct completion of an ordering process. This process is about the delivery and the transportation of the ordered items to the customer. The specific transportation depends on the weight and size of the order at hand. There are three possible situations: truck load, less than truck load, and single package. Below, we report the description of the process from [3]:

After confirmation of a PO, a route guide needs to be prepared and the trailer usage needs to be estimated. The route guide is prepared by determining the trackpoints that are going to be visited during the shipment. The trailer usage is determined by estimating the number of packages for the shipment, where each package has an identifier and a fixed volume of 25, 50, 100 or 200lbs. These operations are performed in parallel by the two tasks Prepare Route Guide and Estimate Trailer Usage. The former task is allocated to the Shipment Planner with the shortest work queue, while the latter task is allocated to the Shipment Planner that was allocated an instance of this task the longest time ago. If either task takes too long, a time out is triggered, which leads to the cancelation of the PO and the termination of the overall process. This timer is set to five days for a PO with one line item and is increased by one day for each additional line item. This calculation is performed by the automated task Calculate Carrier Timeout, which is assigned to a codelet to perform the required additions. If both tasks Prepare Route Guide and Estimate Trailer Usage are completed in time, a Supply Officer can perform task Prepare Transportation Quote, by establishing the shipment cost based on the number of packages and on the total volume of the freight, and by assigning a shipment number to the order number. Once a Supply Officer has chosen to perform this task they have the privilege to reallocate it to someone else without loss of the work performed thus far, and to suspend and resume working on the task. In addition, the system automatically starts the task for a Supply Officer once they have committed themselves to performing it. After task Prepare Transportation Quote, based on the total volume of the freight and on the number of packages, a distinction is made among shipments that require a full truck load (total volume greater than or equal to 10.000lbs), those that require less than a truck load (total volume less than 10.000lbs and more than 1 package) and those that simply concern a single package (total volume less than 10.000lbs). For shipments that require a full truck load, Client Liaisons from the OD try to arrange a Pickup appointment and a Delivery appointment, by specifying the location for pickup/delivery and any specific instructions. The Client Liaisons associated with these two tasks should be different. It is possible that only one of these or even none of these appointments is made before a Senior Supply Officer holding a Masters in Supply Chain and Logistics Management decides to create a Shipment Information document. The Shipment Information document is used by the Senior Supply Officer to specify an authorization code and a consignee number for the shipment number. After the creation of this document, any missing appointments are made, though this time a Warehouse Officer takes charge of arranging a Pickup appointment and a Supply Officer takes care of arranging a Delivery appointment, and there are subsequent opportunities to modify them until a Warehouse Admin Officer produces a Shipment Notice after which the freight can actually be picked up from the Warehouse. Modifications of Pickup appointments are handled by a Warehouse Officer while modifications of Delivery appointments are taken care of by a Supply Officer. When the shipment consists of more than one package but a dedicated truck is not required, a Warehouse Officer arranges a Pickup appointment and a Client Liaison tries to arrange a Delivery appointment. Afterwards, a Senior Supply Officer, who holds a Bachelors in Supply Chain and Logistics Management, creates a Bill of Lading, which, similar to the Shipment Information document, requires the specification of an authorization code and a consignee number. If no Delivery appointment was made prior, a Supply Officer takes care of this and the remainder of the process is the same as for a shipment that requires a dedicated truck. For shipments consisting of a single package the process is straightforward. All that needs to be done is for a Supply Officer — one that has the most experience in performing this particular task (identified using the "Round Robin by Experience" allocation strategy) — to create a Motor Carrier Pickup manifest. This is done by specifying only an authorization code. Afterwards a Shipment Notice is produced by a Warehouse Admin Officer and the freight is ready for pickup. A Shipment Notice provides a summary of the shipment which includes the shipment number, the order number, the number of packages, the pickup and delivery appointments and a variable indicating whether the shipment is a full truck load. The Warehouse Admin Officer has to indicate when the freight loading on the carrier's truck started and completed and provide the details of the driver (number and name), the delivery in-

structions and a deadline for the claims which will be used later on in sub-process Freight Delivered. Delivery instructions contain textual instructions, the delivery date and the delivery location (the latter being retrieved from the Route Guide).”

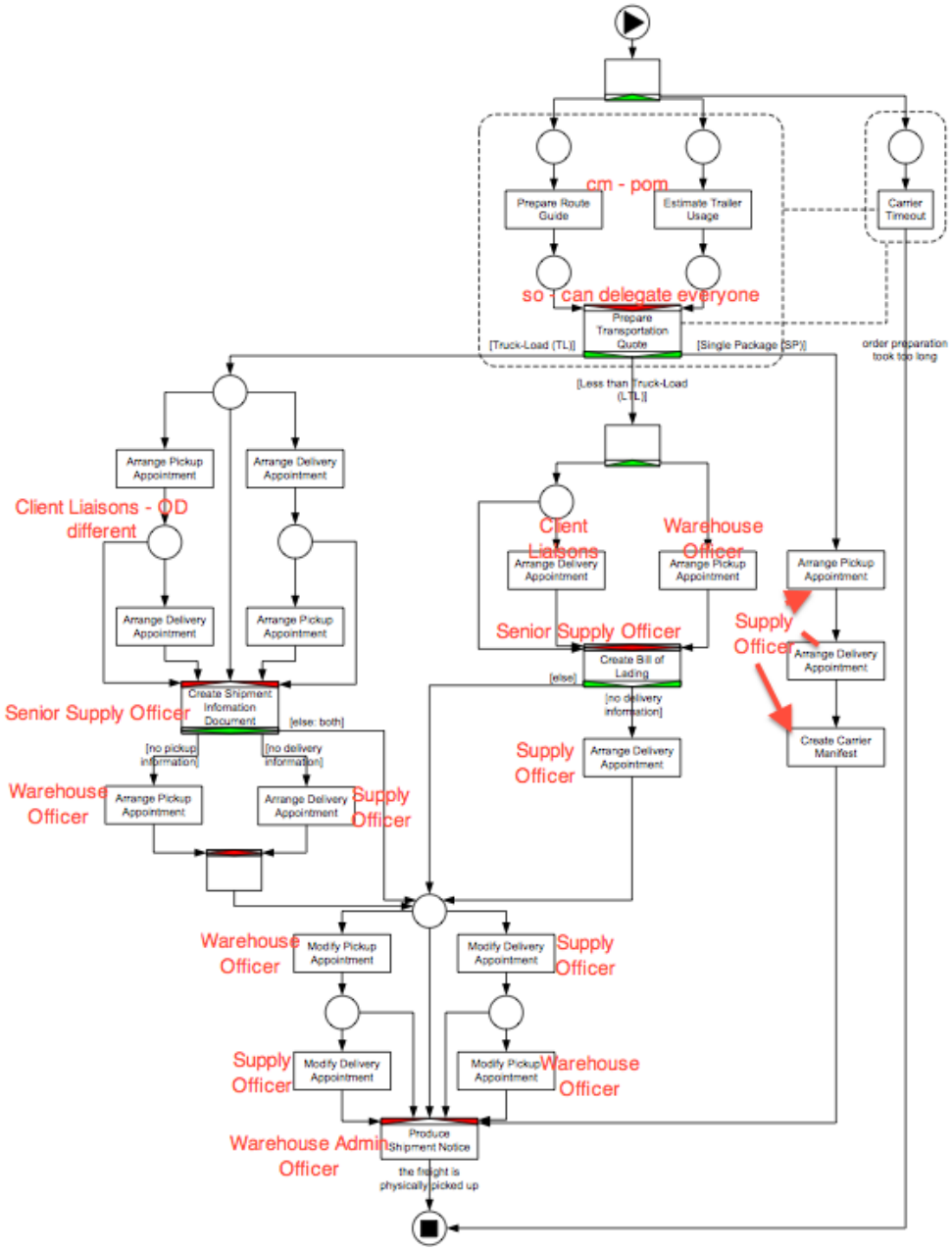

Figure 4: Carrier appointment process in YAWL

Figure 4 shows the carrier appointment business process as a workflow in YAWL. The process also highlights the performer who is assigned to the tasks. This is an example of a complex protocol, which has different partially overlapping paths and a complex sequencing among activities. This model attempts to accommodate a large number of variants in a language that makes it difficult to specify flexible workflows. As a result, the specification becomes unwieldy and unmanageable, too complex to be extensible or even comprehensible [2, 1].

Table 2 shows the carrier appointment business process as an Azzurra protocol. When the protocol

protocol CarrierAppointment (**context** org, pom: PurchaseOrderManager, supplyOff: SupplyOfficer, clLiaison: ClientLiaison) {
ag-variables: whAdminOff: WarehouseAdminOfficer;
commitments:
init $\rightarrow\!\!\!\!\rightarrow$ C_1:C(pom, supplyOff, $\top$, RouteGuidePrepared $\wedge$ TrailerUsageEstimated)
TrailerUsageEstimated $\rightarrow\!\!\!\!\rightarrow$ C_2:C(supplyOff, pom, $\top$, TrQuotePrepared)
TrQuotePrepared [TLNeeded] $\rightarrow\!\!\!\!\rightarrow$ C_3:C(supplyOff, whAdminOff, $\top$, fulfil-p(Truck-Load(org, whAdminOff, clLiaison))
TrQuotePrepared [LTLNeeded] $\rightarrow\!\!\!\!\rightarrow$ C_4:C*(supplyOff, whAdminOff, $\top$, fulfil-p(Less-Truck-Load(org, whAdminOff, clLiaison))
TrQuotePrepared [SPNeeded] $\rightarrow\!\!\!\!\rightarrow$ C_5:C*(supplyOff, whAdminOff, $\top$, fulfil-p(Single-Package(org, supplyOff))
deadline(C_1, 5 days)
can-deleg-ret-resp(C_1)
can-deleg-no-resp(C_2)
kb:
implies(TLNeeded, TrailerUsageEstimated)
implies(LTLNeeded, TrailerUsageEstimated)
implies(SPNeeded, TrailerUsageEstimated)
mut-excl(TLNeeded, LTLNeeded, SPNeeded)
}

Table 2: Carrier appointment process in Azzurra

CarrierAppointment is instantiated, an instance of commitment C_1 shall be created from the purchase order manager to the supply officer (both parameters of the procotol, thus, bounded at instantiation time); the commitment is about preparing the route guide and estimating the trailer usage for the order. These documents need be prepared within five days. The product order manager can delegate the commitment, but, while doing so, retains its accountability for it. In turn, the supply officer commits to prepare the transportation quote, once the trailer usage has been estimated (C_2). Once the trailer quote has been prepared, and depending on the truck load, the supply officer commits to fulfill one sub-protocol: Truck-Load when the order requires a dedicated truck, Less-Truck-Load when the order requires part of the truck, and Single-Package when there is only a package to deliver. Notice how the mut-excl constraint is used to ensure that only one sub-protocol is instantiated. Figure 5 shows the graphical notation of the protocol. Notice the usage of sub-protocols (rectangles with a folder icon); in this figure, sub-protocols are collapsed. However, the support tool for Azzurra enables expanding those protocols.

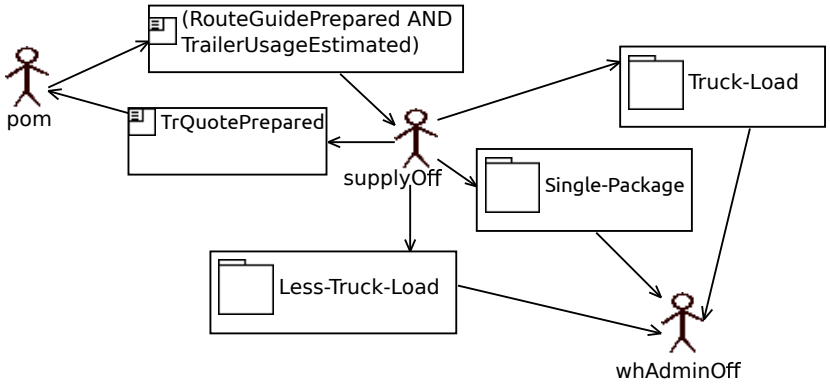

Figure 5: Carrier appointment process in Azzurra (graphical representation)

Table 3 illustrates the Azzurra protocol for the Truck-Load sub-process. Upon instantiation, instantiated,

protocol Truck-Load (**context** org, whAdminOff: WarehouseAdminOfficer, clLiaison1: ClientLiaison) {
ag-variables: clLiaison2: ClientLiaison, warehouseOff: WarehouseOfficer, srSupplyOff: SeniorSupplyOfficer;
commitments:
init $\twoheadrightarrow$ C_1:C(clLiaison1, whAdminOff, $\top$, PickupArranged)
PickupArranged $\twoheadrightarrow$ C_2:C(clLiaison2, whAdminOff, $\top$, DeliveryArranged)
init $\twoheadrightarrow_{\leq 1h}$ C_3:C*(srSupplyOff, warehouseOff, PickupArranged $\wedge$ DeliveryArranged, ShipmentInfoDocCreated(version))
ShipmentInfoDocUpdated(version) $\twoheadrightarrow$ C_4:C(warehouseOff, whAdminOff, $\top$, ShipmentNoticeProduced(newVersion))
sep-duties(C_1, C_2)
kb:
implies(ShipmentInfoDocCreated, ShipmentInfoDocUpdated)
implies(PickupModified, ShipmentInfoDocUpdated)
implies(DeliveryModified, ShipmentInfoDocUpdated)
}

Table 3: Specification for the truck-load sub-process in Azzurra

C_1 and C_3 shall be created, by clLiason1 and srSupplyOff, respectively. The latter commitment is made to an agent variable which is not bounded; therefore, the senior supply officer decides upon the warehouse officer to choose. That agent will be bounded to warehouseOff once the commitment is made. Notice that C_3 can actually be created within 30 minutes after the protocol is instantiated; this enables the senior supply officer contacting a warehouse officer. C_3 is a strong commitment (see the "*" annotation); that means that the consequent shall be brought about (shipment information document be created) only after the antecedent is brought about (pickup and delivery have been arranged). Similarly to the ordering protocol, this protocol supports dealing with modifications (here, of the pickup and the delivery details). We exploit the separation of duties between commitments C_1 an C_2 to express that pickup and delivery shall be arranged by different liaisons.

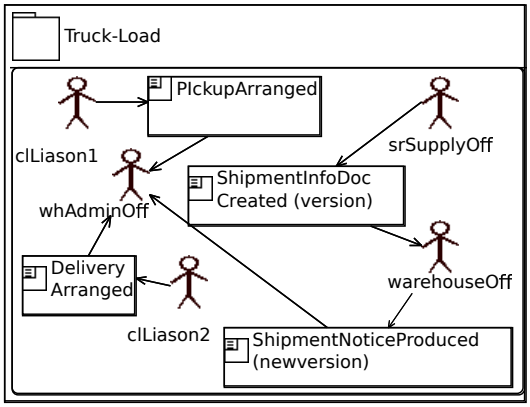

Figure 6: Truck-load sub-process in Azzurra (graphical representation)

Table 4 shows the protocol for the Less-Truck-Load sub-process. The protocol is simpler than the Truck-Load case, as it involves a single liaison. Notice how C_1, C_2, and C_3 represent the fact that delivery and pickup can be performed in any order, and that the bill of lading shall be created after delivery was arranged. Compare this to the YAWL specification in Figure 4, which requires a complicated control-flow structure to represent the same business pattern. Figure 7 shows the graphical representation of the protocol.

protocol Less-Truck-Load (**context** org, whAdminOff: WarehouseAdminOfficer, clLiaison1: ClientLiaison) {
ag-variables: srSupplyOff: SeniorSupplyOfficer, warehouseOff: WarehouseOfficer;
commitments:
init $\twoheadrightarrow$ C_1:C(clLiaison1, srSupplyOff, $\top$, DeliveryArranged)
init $\twoheadrightarrow$ C_2:C(warehouseOff, srSupplyOff, $\top$, PickupArranged)
PickupArranged $\twoheadrightarrow$ C_3:C*(srSupplyOff, warehouseOff, $\top$, DeliveryArranged $\wedge$ BillOfLadingCreated(version))
BillOfLadingUpdated(version) $\twoheadrightarrow$ C_4:C(warehouseOff, whAdminOff, $\top$, ShipmentNoticeProduced(newVersion))
kb:
implies(BillOfLadingCreated, BillOfLadingUpdated)
implies(PickupModified, BillOfLadingUpdated)
implies(DeliveryModified, BillOfLadingUpdated)
}

Table 4: Specification for the less-truck-load sub-process in Azzurra

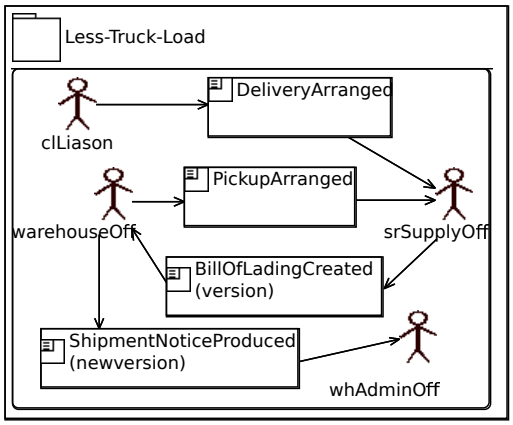

Figure 7: Less-truck-load sub-process in Azzurra (graphical representation)

protocol Single Package (**context** org, supplyOff: SupplyOfficer) {
ag-variables: warehouseOff: WarehouseOfficer;
commitments:
init $\twoheadrightarrow$ C_1:C(supplyOff, warehouseOff, $\top$, PickupArranged $\wedge$ DeliveryArranged $\wedge$ CarrierManifestCreated)
CarrierManifestCreated $\twoheadrightarrow$ C_2:C(warehouseOff, supplyOff, $\top$, ShipmentNoticeProduced)

Table 5: Specification for the single-package sub-process in Azzurra

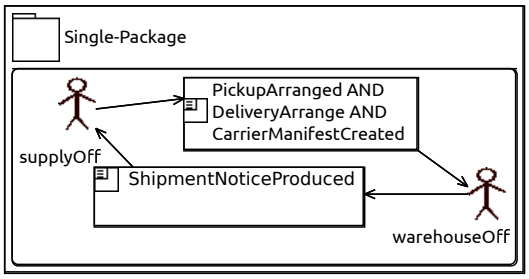

Figure 8: Single-package sub-process in Azzurra (graphical representation)

Table 5 shows the Single-Package protocol. When instantiated, an instance of C_1 shall be created: the

supply officer commits to a warehouse officer that the pickup will be arranged, the delivery will be arranged, and the carrier manifest will be created. Notice that the pickup and delivery details cannot be modified—unlike protocols Truck-Load and Less-Truck-Load—.

References

[1] K. Figl and R. Laue. Cognitive Complexity in Business Process Modeling. In *Proc. of CAiSE'11*, volume 6741 of *LNCS*, pages 452–466. Springer, 2011.

[2] M. La 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*, 7(4):614–629, 2011.

[3] M.L. Rosa, S. Clemens, A. Hofstede, and N. Russell. Appendix a the order fulfillment process model. *Modern Business Process Automation*, pages 599–616, 2010.

[4] M. P. Singh. An Ontology for Commitments in Multiagent Systems: Toward a Unification of Normative Concepts. *Artificial Intelligence and Law*, 7(1):97–113, 1999.

[5] W. M. P. van der Aalst, M. Pesic, and H. Schonenberg. Declarative Workflows: Balancing between Flexibility and Support. *Computer Science-Research and Development*, 23(2):99–113, 2009.

[6] W.M.P. Van Der Aalst and A.H.M. Ter Hofstede. Yawl: yet another workflow language. *Information Systems*, 30(4):245–275, 2005.

A Syntax of Azzurra

We present the syntax of the Azzurra language through the EBNF in Table 6. Terminal symbols are in bold, non-terminals in italics. We illustrate our syntax in Table 7 through the following scenario from the literature:

Example 1 (Fracture treatment [5]) *The patient is initially examined by a specialist. In case of a fracture, x-rays are requested. Specific treatments are applied depending on the diagnosis: sling, fixation, surgery, and cast. The handling doctor is responsible for choosing. Cast and fixation are mutually exclusive. If no fracture is found, a sling has to be made. Patients who undergo surgery are advised to execute rehabilitation. Medications can be provided and additional x-rays can be performed if needed.* □

Protocol signature (1,3). Each protocol (1) has an identifier p_{id} and a set of parameters (3): a context agent variable (the arbiter for disputes between the debtor and the creditor in a commitment [4]), and a set of agent variables associated with specific roles. These variables indicate those agents that have to be bounded to certain roles in order to instantiate the protocol. In Table 7, the unique identifier is Treatment, the context agent is hospital h, and there are two agent variables: patient p and specialist sp.

Protocol specification (2). It includes a set of typed agent variables (their type is a role), a set of commitment classes, a set of role conflicts (optional), and a knowledge base that defines semantic relations between atomic propositions. A role conflict is a binary predicate role-confl: an agent cannot play two roles in the same protocol instance. Role conflicts have a unique identifier; among other reasons, such identifier enables specifying compensations for violated role constraints.

Agent variables (2,4). We support agent variables that are unbounded at protocol instantiation time. They are bounded when an instance of a commitment where they appear is created, and, as an additional

Table 6: Syntax of Azzurra; terminals in bold, non-terminals in italics

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

effect, the bounded agent adopts the specified role in the protocol instance. Azzurra supports assign-once variables: once an agent is assigned, no other agent can be assigned to that variable. In Table 7, agent variables exists for a rehab centre, a radiologist, an orthopedist, a surgeon, and a nurse.

protocol Treatment (**context** h, p : Patient, sp : Specialist) {
ag-variables: rc : RehabCentre, ra : Radiologist, or : Orthopedist, su : Surgeon, nu : Nurse;
commitments:
init $\twoheadrightarrow$ C_1:C(sp, p, $\top$, Examined $\cdot$ Diagnosed)
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 [$\neg$Refused] $\twoheadrightarrow$ C_7:C(nu, p, $\top$, RcChosen(rc))
RcChosen(rc) $\twoheadrightarrow$ C_8:C(rc, p, $\top$, RehabGiven)
MedPrescribed(m) $\twoheadrightarrow$ C_9:C(nu, sp, $\top$, MedApplied(m))
can-deleg-no-resp(C_3) deadline(C_2, 2h)
kb:
implies(XRayRequested, Diagnosed) implies(NoXRayNeeded, Diagnosed)
implies(MedPrescribed(m), Diagnosed) mutExcl(XRayRequested, NoXRayNeeded) }

Table 7: Azzurra protocol for the fracture treatment scenario

Commitments (5,6). The core of a protocol (5) includes commitment classes and their refinements. Azzurra supports two types of commitment classes (6): C and C*. The former says that, if an instance of event ev happens, and if the precondition prop (if specified) holds, an instance of the commitment class id has to be created by the debtor to the creditor. The latter (C*) requires the debtor to bring about the consequent only after the antecedent has occurred. An optional deadline ($\leq$ *time*) specifies that the commitment should be created within an amount of time since the trigger event occurred (as opposed to instantaneously). Debtor and creditor shall be such that:

- if the agent variable is bounded to agent a, a shall be debtor (or creditor);
- if the agent variable is unbounded, any agent a' can be debtor (or creditor), and a' becomes bounded to the agent variable by making the commitment.

In Table 7, commitment C_1 is triggered when the protocol is instantiated: sp commits to p that he will be Examined and then Diagnosed. C_2 says that, if x-rays are not needed, an orthopedist will commit to sp to make a sling. C_4 shows strong commitments: the specialist commits to assess the fracture only after x-rays have been performed.

Commitment refinements (7). A deadline commits the debtor to bring about the consequent within a certain time after the commitment detachment (when the antecedent occurs). The debtor can be authorized to delegate the commitment, either retaining (can-deleg-ret-resp) or releasing (can-deleg-no-resp) responsibility. The creditor, similarly, can be authorized to assign the commitment, either retaining (can-assign-ret-cred) or releasing (can-assign-no-cred) his credit. The debtor can be authorized to cancel her commitment (can-cancel). Commitment refinements have no identifier, for they are a fine-grained specification of another commitment. In Table 7, the radiologist can delegate C_3, possibly to a colleague, without retaining responsibility.

Protocol refinements (8). They constrain agents in a protocol instance. A maximum number of concurrent commitments (max-per-role) can be specified for an agent playing a certain role. A specialization of it limits the number of instances of a commitment class (max-of-class). Separation of duties (sep-duties) implies that, given two commitment classes, an agent cannot be debtor in instances of both. Separation of duties can be restricted to agents playing a specific role (comm-role-confl). The compensate refinement specifies that, if an instance of the first commitment class is violated, then the debtor of that instance has to create an instance of the second commitment class.

Propositions and events (9,10,13,15). Azzurra supports different proposition types (9): atomic propositions (atom), commitment states (cState), protocol states (pState), binary operators over propositions, negated propositions, propositions between brackets. The binary operators (10) 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. FractAsessed). 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 (13), as they trigger new commitment instances and change the state of existing commitment instances. Four event types are supported:

- the protocol at hand is instantiated (init);
- an atomic proposition becomes true. This includes the occurrence of a state of affairs (e.g., the patient is diagnosed) and the adoption of roles (e.g., jim:Patient);
- the state of a commitment instance changes (see clause (13) below);

- the state of another protocol instance changes (see clause (12) below).

Protocol states (12). Azzurra enables cross-protocol references, e.g., when an agent commits to successfully fulfill a protocol. For the time being, we support a simple language to talk about the state of other protocols. Given a protocol instance p:

- p is initialized (init-p) when all the commitments triggered by event *init* are created;
- p is fulfilled (fulfil-p) when there are no pending commitment instances. We do not distinguish between success and failure of an entire protocol here.

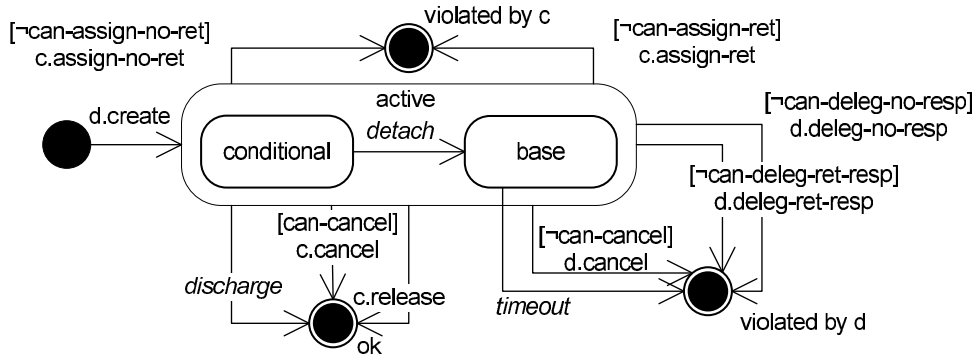

Figure 9: Runtime semantics for a commitment instance made by debtor d to creditor c

Commitment states (13). Propositions and events may denote that a commitment is or has changed to a specific state (as described in Fig. 9). 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 cancelled;
- 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 (14). 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. In Table 7, three states of affairs imply that a diagnosis was done: XRayRequested, NoXRayNeeded, and MedPrescribed. Also, XRayRequested is mutually exclusive with NoXRayNeeded.