Formalizing Commitments Using the Event Calculus

Joost de Kruijff, Hans Weigand

Tilburg University, Tilburg, The Netherlands
{j.c.dekruijff,h.weigand}@uvt.nl

Abstract. Smart Contracts enable the automated execution of exchanges on the blockchain. From an ontological perspective, smart contracts create and automate the fulfillment of social commitments between actors. Whereas traditional deontic logic is used to make a legal determination in contractual multi-actor interactions, we focus on the consequences of these actions resulting from that determination, thereby shifting the focus from monitoring to execution. The interactions between actors and the consequences in terms of commitments have not yet been formalized for smart contracts. The perspective of smart contracts is interesting, since they are considered to be autonomous agents, able to generate automated actions. We use the Event Calculus to formalize logic in order to represent and reason about the effects of these automated actions and the resulting commitments. Since the Event Calculus deals with local events and the consideration of time, this approach enables the uniform representation of commitments, including their operations and reasoning rules about them.

Keywords: Blockchain, Smart Contracts, Commitments, The Event Calculus

1 Introduction

In line with Enterprise Ontology and Business Ontologies like COFRIS [1] and REA [2], we model smart contracts as a bundle of interrelated commitments (together: a contract) among those parties who have signed it. The main objective of a contract for the contract agents is to fulfill a certain goal and to safeguard against undesirable outcomes, together referred to as contract robustness [3]. Contracts that are not robust may lead to transaction costs, expensive conflict resolution, or even a collapse of a transaction as a whole. According to the sociological account of [4], “commitments” are needed to explain a consistent line of activity and come into being when an individual brings in extraneous interests (the example used in this paper is specifically on “side bets”). These “side bets” are often a consequence of the actors' participation in social interactions. A more abstract way of saying the same is that commitments penalize the individual in the case of inconsistent behavior, and that the penalty has its basis in the social community. Importantly, the assumed effect of commitments is consistency in behavior, and this contributes directly to contract robustness. As a result, the role of commitments for decent business transactions is essential. It is not surprising that commitments are basic in UFO-S, the foundational ontology of services. They are also included in UFO-C (ontology of social entities) as a kind of social moments, that is, "types of intentional moments that are committed by social actions (e.g., an interaction composed of the exchange of communicative acts)" [5].

In 1969, [6] provided an elegant way to logically represent changes of the world through actions, captured in a protocol, called the Event Calculus. The Event Calculus enables the uniform representation of commitments, including its operations and reasoning rules about them [7]. Event calculus originates from the Situation Calculus, but there is a conceptual difference between the two: the Situation Calculus deals with global states, whereas the Event Calculus deals with local events and the consideration of time periods. The latter is similar to the structure of a blockchain, whereby transactions that change the state of the ledger (actions) occur sequentially, based on situations (time or condition constraints) as defined in the smart contract. The Event Calculus, can be formalized by means of Horn clauses augmented with negation by failure [6]. Based on the Event Calculus, Singh et al [7] developed a declarative protocol specification by capturing the meaning of actions including intrinsic meanings through commitments. As a result, they defined operations to commit, manipulate and terminate (discharge, delegate, cancel) these commitments, centered on events and fluents. Fluents are properties that may have different values at different time points (states) and the entire lifecycle of a commitment. We consider fulfillments to be a state of a fluent. Events manipulate fluents. A fluent starts to hold after an event occurs that can initiate the fluent. Similarly, it ceases to hold when an event occurs that can terminate the fluent. This paper aims to contribute in two ways. First, we aim to formalize the representation and reasoning of the effects of commitments and automated actions by smart contracts that result from them, with specific regards to creation, manipulation and fulfillment. Second, we introduce flexibility to the process of assigning or delegating control and responsibility roles inside an immutable smart contract, while adding to the robustness of the contract.

2 Conceptual Framework

The conceptual model in this paper builds upon earlier work [2], where we separated the implementation choices for blockchain using three human abilities derived from Enterprise Ontology; *performa*, *informa*, and *forma*. The forma ability concerns the form aspects of communication and information. Production acts at the forma level are datalogical in nature: they store, transmit, copy, destroy, etc. data. The informa ability concerns the content aspects of communication and information. Production acts at the forma level are infological in nature, meaning that they reproduce, deduce, reason, compute, etc. information, abstracting from the form aspect. The performa ability concerns the bringing about of new, original things, directly or indirectly by communication. Communicative acts at the performa level are about evoking or evaluating commitment; these communicative acts are realized at the informa level by means of messages with some propositional content. We previously presented our conceptual model for smart contracts at the informa level, where we emphasized the infological blockchain domain ontology to accommodate COFRIS-related components at the performa level [1]. At the infological layer, the notion of transaction has been refined to three aggregation levels: transaction, event and posting. In this paper we extend our infological ontology by including formalizations for commitments through Event Calculus. Event calculus semantically extends our ontology by offering tools to Commit and manipulate commitments. We first explain and present the Event Calculus extensions for commitments, and then we will map these formalisms to our infological ontology.

2.1 Commitment Lifecycle

In line with [14], new knowledge changes a state (in terms of fluents) by fulfilling (partial) or realizing (full) a commitment *c* over time through transactions on the SL. A commitment here is a conditional business relationship directed from a debtor to a creditor, and can be formalized as *C(debtor, creditor, antecedent, consequent* [7]. The lifecycle of a commitment has been explained by Telang and Singh [8] as follows:

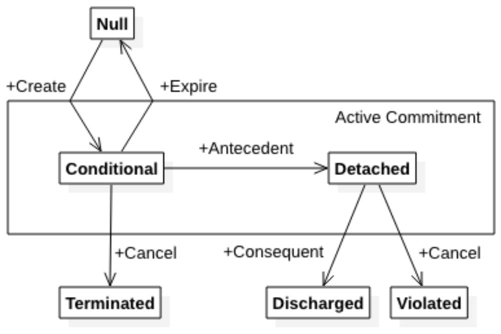

Figure 1. The Commitment Lifecycle by Telang and Singh [8] in UML

A commitment transitions from one state to another through the operations: *Commit*, *detach* (antecedent holds), *discharge* (consequent holds), *cancel*, *assign* and *delegate.* A commitment can be in one of the following states: Null (before it is Committed), Existing (when it is initially Committed), Active (when it is initialized), Expired (when its antecedent becomes forever false, while the commitment was Conditional), Discharged (or Satisfied) (when its consequent is brought about while the commitment was Active, regardless of its antecedent), Violated (when its antecedent has been true but its consequent will forever be false, or if the commitment is cancelled when Detached), Terminated (when cancelled while Conditional or released while Active), or Pending (when suspended while Active). Active has two sub-states: Conditional (when its antecedent is still false) and Detached (when its antecedent has become true). A debtor may Commit, Cancel, Suspend, or Reactivate a commitment. A creditor may Release a debtor from a commitment [8]. Although the commitment lifecycle is the basis for our conceptual model, we do make an important adjustment by adding the option to delay initialization (or activation) of a commitment in order to prevent the existence of active commitments that have no chance to satisfy (falsely pending). For example, a commitment to consider offers on a piece of property, will only activate once offers can be made by buyers to satisfy that commitment. Hence the commitment is made (way) earlier than the property was listed. In this scenario, there is a time gap between the creation and the activation of a commitment.

Actions are realized by means of messages and are a conjunction of subactions. Actions may either be predefined or Committed on-the-fly by participants. Governatori [9] recommends to predefine action logic explicitly for better monitorability of the contract. We introduce *Exchange Commitments* and *Control Commitments* to distinguish between commitments that are defining the economic transaction (are about the resource exchange) like creating, activating and satisfying and those that deal with the conditioning of the contract like delegating or assigning. From a speech act perspective, commitments are a structure to communicate meaning and intent between actors. This structure as presented by [10] distinguishes between a success- and a dispute or failure layer. The success layer contains the set of communicational moves to complete a transaction successfully. Transactions hereby follow the transaction paradigm that state that actors commit and execute actions that result in the creation of new facts. So we consider the action events on this layer to address exchange commitments. Characteristics of the success layer is that the proposition of the transaction is never changed during its lifecycle nor the constraints. On the other hand, the discussion layer (also mentioned as failure and dispute layer) is concerned with the communicative acts for situations where this is or may not be the case. In the event of opportunistic behavior by actors or changed circumstances that require a change to the proposition *after* committing to bring about the original proposition, the transaction should be resolved in a new request or be closed altogether. We introduce control commitments to deal with the conditioning of a contract under these circumstances, like changes to the rules of engagement, delegation- and assignment of actors and violation. Once the transaction diverts from its path towards success, control commitments prescribe what should happen. Key to control commitments is that they allow to change elements of the contract by manipulating commitments through Constraints, without changing the proposition of the transaction itself.

Table 1. Comparison between Communication Layers and Commitments

Communication layers	**Commitments**
Success layer	Exchange commitments
Discussion layer	Control commitments

Constraints consist of Conditions that verify if an Event happened (Happens) or if a commitment holds (HoldsAt). Whereas control commitments are more straightforward, Exchange Commitments between agents are easier to predict when goals of the agents are known, since goals describe the state of the world that an individual agent is motivated to bring about (antecedent or consequent) [11]. We distinguish between the Control and the Responsibility role. The Control role is concerned with the execution of an Event, without social responsibility. Responsibility means social responsibility but not necessarily execution. Roles are not set in stone and may change due to time- or conditional constraints. We consider two role transitions; (1) *Delegation* is concerned with the change of debtor, without changing anything else. Since the creditor and conditions are unchanged, the creditor remains socially responsible for the commitment; he delegates control of execution to the smart contract as in the case of a notary or bank, for example. (2) *Assigning* is concerned with the change of creditor, without changing anything else. Even though the creditor changes, the new creditor becomes participant in bringing about the commitment. This may be the case when a house owner Assigns the ownership transfer to a notary, or in a future situation, to a smart contract.

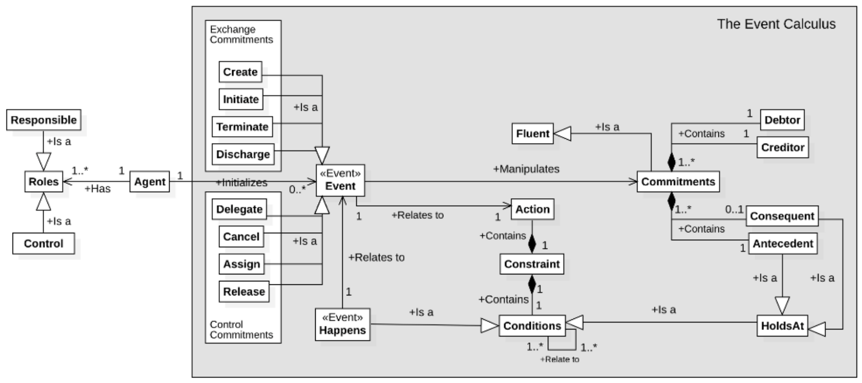

Figure 2. Conceptual Model Commitment Formalization using The Event Calculus

Now that we understand the basic lifecycle of commitments, it is important to map this concept to our infological ontology. Hereby, the concept of Events, and Fluents are mapped to Transfers and Accounts in our infological ontology [12], but the two conceptualizations have a slightly different focus. Whereas a Transfer manipulates agnostic Accounts (through inflow- and outflows), Events manipulate specific Fluents through Actions, namely Commitments, by specific operations from the commitment lifecycle. On the other hand, in the infological model a Transfer is conceived as Inflow and Outflow. So to combine the two models, we have to restrict Transfers to specific commitment-manipulating actions and we have to specify these actions in terms of Inflow and Outflow. The main concern with regards to reasoning about Transfers in smart contracts, is that so far, there is a lack of standards for Rules of Engagement. The Clauses and Defaults that govern the Transfers can be anything. By mapping the Rules of Engagement to Event Calculus axioms, the rules become a sound axiom system. A distinction must be made between the general Event Calculus axioms, generic Commitment axioms and contract-specific rules. For example, it is a generic Commitment axiom that only a debtor Agent is allowed to discharge a commitment. Further, by allowing preconditions to be associated with the Initiation and Termination of properties, different commitments can be associated with communicative acts to model the communications among Agents more concretely.

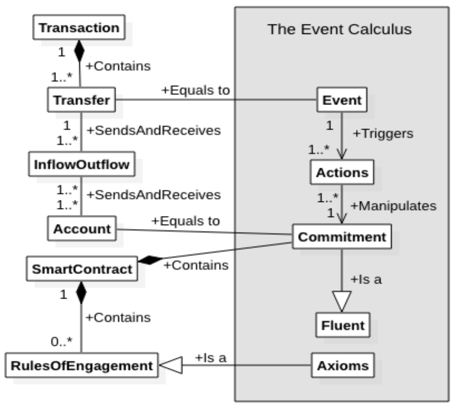

Figure 3. Mapping Infology to The Event Calculus

3 Manipulating commitments using the Event Calculus

The Event Calculus uses various predicates to reason about events. Based on the predicates and axioms presented in [7] [8]. We have modified the notation to *e* for events and *c* for commitments. The symbol ← denotes implication, ∧ denotes conjunction and ~ denotes negation by failure. The time points are ordered by the < relation, which is defined to be transitive and asymmetric. We write a commitment as *C(x, y, p, q)* where *x* is the debtor, *y* is the creditor and *p* and *q* are the antecedent and consequent respectively. When *c* is a commitment, *debtor(c)* yields the *x*, and similar to the other roles Before we explain the axioms that are required to create and manipulate commitments, it is important to distinguish between the various commitment types that have been identified by [7] used in this paper. Base commitments (BC) written as *BC(x, y, p)* are commitments from debtor *x* to a creditor *y* to satisfy condition *p*. Condition *p* does not involve other fluents or commitments, written as *BC(x, y, p).* This type of commitment is also known as an unconditional commitment. On the other hand, conditional commitments are written as *CC(x, y, p, q)* whereby debtor *x* will bring about condition *q* to creditor *y*, once condition *p* is satisfied. In contrast to a BC, the behavior of a CC is slightly more complex. In line with the state knowledge update paradigm [14], a CC is terminated and subsequently reinitiated. In line with [7], we then implement Colombetti's [15] definitions of conditional commitments, where a CC resolves into a new BC upon the realization of *p*. This new BC is committed to satisfy *q.* So when a CC holds and an event happens that fulfils *p*, the CC is terminated and a new BC being created. In this paper, we will not touch persistent commitments (*PC(x, y, G(p))*). Table 2 summarizes the formalization rules and commitment type used in our protocol run. Commitments denoted as *c* can be either a BC or a CC.

Table 2. EC Rules

Rule	**Explanation**	**EC Notation**	**Type**
r1	Creation of a commitment that is not activated during the create event.	*Commit(e, x, c) ← Happens(e, t)*	Exchange
r2	Activation of a commitment or the activation of a commitment that is activated in the same event	*Activate (e, c, t) ← Happens(e, t) ^ Commit(e, x, c)*	Exchange
r3	Termination of a BC that is fulfilled	*Satisfy(e, BC(x, y p), t) ← Happens(e, t) ^ Activate(e, bc(x, y, p), t) ^ Discharge(e, x, BC(x, y, p))*	Exchange
r4	Termination of a CC that resolves in a BC to provide q in a later event	*Activate (e, BC(x, y, q), t) ← Happens(e, t) ∧ Commit(e, x, BC(x, y, q))* *Satisfy(e, CC(x, y, p, q), t) ← Happens(e, t) ∧ HoldsAt (CC(x, y, p, q)) ^ Activate(e, CC(x, y, p, q), t))*	Exchange
r5	Termination of a CC that resolves in a BC to provide q that is activated in the same event of satisfying the CC	*Commit(e, x, BC(x, y, q)) ← Happens(e, t)* *Satisfy(e, CC(x, y p, q), t) ← Happens(e, t) ∧ HoldsAt (CC(x, y, p, q)) ∧ Activate(e, CC(x, y, p, q), t))*	Exchange
r6	The delegate operation that replaces the debtor of the commitment with agent z	*Commit(e, z, CC(z, y, p, q)) ← Happens(e, t) ∧ Delegate(e, x, z, CC(z, y, p, q))* *Cancel(e, x, CC(x, y, p, q)) ← Happens(e, t) ∧ Delegate(e, x, z, CC(x, y, p, q))*	Control
r7	The assign operation that replaces the creditor of the commitment with agent z	*Commit(e, x, cc(x, z, p, q)) ← Happens(e, t) ∧ Assign(e, z, z, cc(x, y, p, q))* *Release(e, y, cc(x, y, p, q)) ← Happens(e, t) ∧ Assign(e, y, z, cc(x, y, p, q))*	Control

The rules presented in table 2. İmplement rule templates that are (still under development) and out of scope for this paper. In that context, r_1 should be considered as a rule that compounds multiple rules in our workflow to relate commitments as the evolve. For example, the commitment to Relocate has a direct relation with the commitment to ConsiderOffers.

4 Application: The Real Estate Example

We now apply the CBSC framework to a basic real-estate transaction - often mentioned as an important application area for smart contracts [16]. *BUY* represents the buyer (or debtor agent). *SEL* represents the seller (or creditor agent) and *AG* represents a real estate agent.

Since this process is standardized and regulated in most countries, there is only one protocol run possible. We assume that only one action can occur at one time point. Hence, we are not concerned with concurrent events. The frame problem [6] is handled through circumscription as shown by [13]. Through circumscription, the set of *Activates*, *Satisfies* and *Release* clauses is kept to minimize unexpected effects. The minimization of *Happens* leads to a minimal number of unexpected events. table 3 shows the EC events e1..e5 that correspond to interface messages of the smart contract. Since this property event cycle is heavily regulated, the order of events is rather standard and does not provide room for change. To name the events, we have chosen to uppercase the second main action words, like listProperty, where the first word is a fluent. The next step is to convert the chain of events to the EC. *Commit(e, x, c)* establishes commitment *c*, in our interpretation *BC or CC.* When event *e* is performed, the commitment *c* or a state (fluent) *p* is *initiated*. *HoldsAt* explains which states (fluents) hold at a given time point, and *Happens* defines a predicate relation between events possibly surrounded by conditions and times [17].

Table 3. Protocol Run Events

Event	Event Description	Event Notation	P/Q	Detail
e_1	The event where the seller **decides** to relocate to another location, which would result in the sale of his/her property	DecideToRelocate(a)	f_1	Decide
e_2	The event where the seller **lists** his/her property and starts considering offers from buyers	ListProperty(a, m)	f_2	List
e_3	The event where the buyer **offers** to buy the property from the seller for a certain amount	MakeOffer(a, m)	f_3	Offer
e_4	The event where the seller **accepts** the offer made by the buyer. In this example, the number of offers is not relevant	AcceptOffer(a, m)	f_4	Accept
e_5	The event where the seller and buyer **sign** the agreement	SignContract(a, m)	f_5	Sign

Table 4 shows the protocol run for the first five events of a generic real estate transaction, including the agent that controls the execution and the one that is responsible. We have modified the protocol as such that we could illustrate all rules presented in table 2.

Table 4. Protocol run

Event	Time	P/Q	Rule	EC	C	Title	Control	Responsible
e1	t1.1	f1	r2	Commit	bc1	Relocate	SEL	SEL
	t1.2		r2	Activate	bc1	Relocate	SEL	SEL
e2	t2.1	f2	r3	Satisfy	bc1	Relocate	SEL	SEL
	t2.2		r1	Commit	cc2	Consider	SEL	SEL
e3	t3.1	f3	r1	Commit	cc3	Buy	BUY	BUY
	t3.2		r2	Activate	cc2	Consider	SEL	SEL
	t3.3		r6	Delegate	cc3	Buy	SC	BUY
	t3.4		r2	Activate	cc3	Buy	SC	BUY
e4	t4.1	f4	r5	Satisfy	cc2	Consider	SEL	SEL
	t4.2		r5	Commit	bc4	Consider	SEL	SEL
	t4.3		r2	Activate	bc4	Consider	SEL	SEL
	t4.4		r3	Satisfy	bc4	Consider	SEL	SEL
	t4.5		r1	Commit	cc5	Sell	SEL	SEL
	t4.6		r7	Assign	cc5	Sell	SC	SEL
	t4.7		r2	Activate	cc5	Sell	SC	SEL
	t4.8		r4	Satisfy	cc5	Sell	SC	SEL
	t4.9		r4	Commit	bc6	Sell	SC	SEL
	t4.10		r4	Activate	bc6	Sell	SC	SEL
	t4.11		r4	Satisfy	cc3	Buy	SC	BUY
	t4.12		r4	Commit	bc7	Buy	SC	BUY
	t4.13		r4	Activate	bc7	Buy	SC	BUY
e5	t5.1	f5	r5	Satisfy	bc6	Sell	SC	SEL
	t5.2		r2	Satisfy	bc7	Buy	SC	BUY
	t5.3		r2	Commit	cc8	Transfer	BUY	BUY
	t5.4		r2	Commit	cc9	Vacate	SEL	SEL

For each event and subsequent time point in table, we can apply the rules as defined in table 4. We will illustrate some examples. The example starts with the (arbitrary) decision by the seller to commit to relocate for any reason. Since the seller commits to his/herself in this instance, the seller represents both x and y. It is important to note that we do not imply R_1 here, since the commitment and its activation both happen in e1. We could write this commitment conforming to R_2 in two ways:

Activate(DecideToRelocate(Mainstreet 1), Relocate(SEL, SEL, Decide(Mainstreet 1)), t $_{1.2}$*) ← Happens(DecideToRelocate(Mainstreet 1), t* $_{1.1}$*) ∧ Commit(DecideToRelocate(Mainstreet 1), SEL, Relocate(SEL, SEL, Decide(Mainstreet 1)))*

Or simplified via the shortcuts provided in table 3 and 4. Hereby, *a* represents the asset (e.g. Mainstreet 1) and *m* the amount paid (e.g. 100.000 USD).

Activate(e_1 *(Mainstreet 1),* c_1 *(SEL, BUY,* F_1*),* $t_{1.2}$*) ← Happens(* e_1*(Mainstreet 1),* $t_{1.1}$*) ∧ Commit(* e_2*, SEL,* c_1 *(SEL, SEL,* f_1*(a, m)))*

We prefer the second method using short codes as a matter of convenience. c_1 is satisfied by listing the property. This does not mean that the seller is relocated yet, but all he/she could do after committing to relocate was to list the property. The *list* fluent in e_2 commits the seller to consider offers. The Consider commitment (c_2) imposes r_1 and is not yet activated, since there are no offers yet to consider.

> $Satisfy(e_2(a, m), c_1, t_{2.1}) \leftarrow Happens(e_2, t_{2.1}) \land Activate(e_1, c_1, t_{1.2}) \land Discharge(e_2, SEL, c_1(SEL, BUY, f_2))$
>
> $Commit(e_2, SEL, c_2 (SEL, BUY, f_2(a, m))) \leftarrow Happens(e_2, t_{2.2})$

The Buy and Sell commitments stretch from e_3 (offer) to e_5 (sign) and comply to r_4, whereby the conditional commitments resolves as *p* (accept) holds into a new base commitment to provide *q* (sign). The Buy commitment is created in an event (e_3) where it could not be activated (only from e_4). In contrast to the Consider conditional commitment which implies r_5, *q* is only brought about in a later event at $t_{5.1}$ and $t_{5.2}$. The Buy commitment evolves as follows:

> $Activate (e_4, bc_7(BUY, SEL, f_5(a, m)), t_{4.13}) \leftarrow Happens(e_4, t_{4.13}) \land Commit(e_4, BUY, bc_7(BUY, sell, f_5(a,m)))$
>
> $Satisfy(e_4, cc_3(BUY, SEL, f_4(a,m), f_5(a,m)), t_{4.11}) \leftarrow Happens(e_4, t_{4.4}) \land HoldsAt (cc_3(BUY, SEL, f_4(a,m), f_5(a,m))) \land Activate (e_3, cc_3 (BUY, SEL, f_4(a,m), f_5(a,m)), t_{3.4}))$
>
> $Activate (e_3, cc_3(BUY, SEL, f_4(a,m), f_5(a,m)), t_{3.4}) \leftarrow Happens(e_3, t_{3.1}) \land Commit(e_3, BUY, cc_3(BUY, SEL, f_4(a,m), f_5(a,m)))$

The Buy commitment also delegates control to the smart contract conforming to r_6 by changing *x* to *z*. Please note that *x* remains responsible.

> $Commit(e_3, SC, cc_3(SC, SEL, f_4(a,m), f_5(a,m))) \leftarrow Happens(e_3, t_{3.3}) \land Delegate(e_3, BUY, SEC, cc_3(SC, SEL, f_4(a,m), f_5(a,m)))$
>
> $Cancel(e, x, cc(x, y, f4(a,m), f5(a,m))) \leftarrow Happens(e_3, t_{3.3}) \land Delegate(BUY, SC, cc_3(BUY, SEL, f4(a,m), f5(a,m)))$

The process for the assignment operation for the Sell commitment is similar to the delegate operation but conforms to r_7 instead of r_6 to change the creditor agent from *y* to *z*.

5 Conclusion

This paper introduced a commitment based formalization approach towards smart contracts using the Event Calculus. The concept of commitment based smart contracts builds on the idea of expressing multi-agent transaction through the lens of smart contracts and commitments, while delegating and assigning action execution responsibility to smart contracts as (semi)autonomous agents. Hereby, the scope of agents is extended from agents as 'human' to be 'human' and 'non-human'. We believe that commitments can be created, activated and satisfied at different time points across a (business) transaction. In addition, we have added formalisms to change the role that actors play during a transaction. We think that these additions to the commitment lifecycle are useful to apply commitment based approaches towards smart contracts and could be further extended to state monitoring mechanisms, partial fulfillments and approaches towards the (deontic) concept of contract violation.

References

1. H. Weigand, I. Blums, and J. de Kruijff, "Shared Ledger Accounting - Implementing the Economic Exchange pattern" Proc. CAiSE 2018, pp. 342–356, 2018.
2. J. de Kruijff and H. Weigand, "Understanding the Blockchain Using Enterprise Ontology", Advanced Information Systems Engineering, 29th International Conference, CAiSE 2017, Essen, Germany, June 12-16, 2017, Proceedings," pp. 29–43, 2017.
3. M. P. Singh and A. K. Chopra, "Violable Contracts and Governance for Blockchain Applications," 2018.
4. A. K. Chopra *et al.*, "Agent-Oriented Software Engineering XI, 11th International Workshop, AOSE 2010, Toronto, Canada, May 10-11, 2010, Revised Selected Papers," pp. 17–36, 2011
5. Nardi - Guarini
6. J. McCarthy and P. J. Hayes, "Readings in Artificial Intelligence," *Chapter 5 Adv Top*, pp. 431–450, 1981.
7. P. Yolum and M. P. Singh, "Reasoning about Commitments in the Event Calculus: An Approach for Specifying and Executing Protocols," *Ann Math Artif Intel*, vol. 42, no. 1–3, pp. 227–253, 2004.
8. P. R. Telang, M. P. Singh, and N. Yorke-Smith, "Programming Multi-Agent Systems, 9th International Workshop, ProMAS 2011, Taipei, Taiwan, May 3, 2011, Revised Selected Papers," pp. 22–37, 2012.
9. G. Governatori, "Representing Business Contracts in RuleML," *Int J Coop Inf Syst*, vol. 14, no. 02n03, pp. 181–216, 2005.
10. Reijswoud, V.E. van, (1996) The Structure of Business Communication: Theory Model and Application. PhD Thesis, Delft University of Technology, Delft
11. P. R. Telang, M. P. Singh, and N. Yorke-Smith, "A Coupled Operational Semantics for Goals and Commitments", *Journal of Artificial Intelligence Research 65 (2019) 31–85*

12. J. de Kruijff and H. Weigand, “Ontologies for Commitment-Based Smart Contracts” In: Proc. OTM 2017, Rhodes, Greece, October 23-27, 2017, Part II,” pp. 383–398, 2017.
13. M. Shanahan, “An abductive event calculus planner,” *J Log Program*, vol. 44, no. 1–3, pp. 207–240, 2000.
14. R. Kowalski and M. Sergot, “A logic-based calculus of events,” *New Generat Comput*, vol. 4, no. 1, pp. 67–95, 1986.
15. N. Foara and M. Colombetti “A Commitment Based Approach to Agent Communication” *Appl Artif Intell*, vol. 18, no. 9–10, pp. 853–866, 2004.
16. I. Karamitsos, M. Papadaki, N.B, Al Barghuthi “Design of the Blockchain Smart Contract ”, *Journal of Information Security 09(03)*:177-190, January 2018
17. A. Paschke, “ECA-RuleML: An Approach combining ECA Rules with temporal interval-based KR Event/Action Logics and Transactional Update Logics” *ECA-RuleML Proposal for “RuleML Reaction Rules Technical Goup”* – Nov. 2005