Introducing CommitRuleML for Smart Contracts

Joost de Kruijff, Hans Weigand,

Tilburg University, P.O. Box 90153
5000 LE Tilburg, The Netherlands

j.c.dekruijff@uvt.nl, h.weigand@uvt.nl

Abstract.
This short paper takes a closer look at the Event Calculus and the formalisms for representing and reasoning about the effects of automated actions and the commitments that result from them. Since the Event Calculus deals with local events and the consideration of time, it enables the uniform representation of basic-, conditional and persistent commitments, including its operations and reasoning rules about them. Due to the similarities between event, action and state processing and reaction rules, we extend KR ReactionRuleML to emphasize syntax and semantics relevant for commitments, called CommitRuleML. CommitRuleML is used to define commitment-based smart contracts (CBSCs) as contracts that logically (through the Event Calculus) define events (on), conditions (if) and actions (do) in an executable language. A simplified running example illustrates how commitments evolve and fulfill over time.

Keywords: Event Calculus, Smart Contracts, KR ReactionRuleML, CommitRuleML

Introduction

Smart contracts are considered to be intelligent as they have the ability to execute logic by themselves. They act as a computer program that both expresses the contents of a contractual agreement and operates the implementation of that content using blockchain technology, using triggers provided by the users or extracted from the environment [1]. Smart contracts automatically and always execute pre-defined actions when certain conditions are met. Therefore, it is pivotal to have a thorough understanding of the effects of these automated actions. This short paper takes a closer look at the formalisms for representing and reasoning about the effects of these automated actions and the commitments that result from them. The longer-term research objective is to develop a smart contract language that is as transparent as possible about these commitments and effects.

In 1969, [2] provided an elegant way, called the Event Calculus, to logically represent changes of the world through actions, captured in a protocol. Event calculus enables the uniform representation of commitments, including its operations and reasoning rules about them [3]. Event calculus originates from the Situation Calculus, although the main difference between the two is conceptual: 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 with 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. Event calculus, can be formalized by means of Horn clauses augmented with negation by failure [4]. In search for a logical representation for smart contracts, we formalize smart contracts using the Event Calculus. Previously we conceptually introduced commitment-based smart contracts (CBSC) [5] using ReactionRuleML. CBSC are smart contracts that logically (through the Event Calculus) define events (on), conditions (if) and actions (do). as commitments in an executable language. In this short paper, we provide an introduction into CommitRuleML through extending KR ReactionRuleML. We present a simplified

running example operated by CommitRuleML syntax and semantics, illustrating a real estate sales transaction. Hereby, (1) a seller lists a property, (2) receives offers on that property and, once accepted by the seller, (3) buyer and seller reach a verbal agreement. Once the purchase agreement is signed (4), the seller (5) receives the payment in order to (6) transfer ownership of the property to the buyer.

Event Calculus
The interaction between a buyer and a seller is viewed as a multiagent system (MAS). Commitments between MAS agents are an important basis for organizing their interactions [3]. Since humans are opportunistic beings, it is important to look at the events that create and/or manipulate commitments over time. In alignment with [5], each of these events is considered to be an update which adds new knowledge (to the ledger), starting from an initially empty knowledge base. In the context of CBSC, these events equal commitments between a buyer and seller that change the state of the smart contract. Updates are additive in that they add but do not delete information about events. That a relationship no longer holds is represented by adding information which implies the end of the time period. Knowledge is added to the state (through fluents) by fulfilling a commitment *c* over time through transactions. According to [3], three types of commitments exist:

- **Base commitments**; a commitment BC(*x, y, p*) from debtor *x* to a creditor *y* to satisfy condition *p*. Condition *p* does not involve other fluents or commitments.
- **Conditional commitments**; a commitment CC(x, y, p, q) whereby debtor *x* will bring about condition *q* to creditor *y*, once condition *p* is satisfied
- **Persistent commitments**; a commitment PC(*x, y*, *G*(*p*)) from debtor *x* to creditor *y* to ensure that condition *p* holds on all future time points.

We have modified the original notation of commitment C by distinguishing base commitments (BC) and persistent commitments (PC), originally only denoted as C. Variable *x* represents the debtor agent and *y* represents the creditor agent. A fluent *p* describes the state the debtor needs to satisfy in order to fulfil a commitment, vice versa for *q* for the creditor. G defines the 'always' operator as explained in temporal knowledge. We focus on base- and conditional commitments at first for simplicity reasons.

Example
The roles, fluents and commitments in our simplified running example can be defined as follows:

Roles:

- *B* represents the buyer
- *S* represents the seller

Domain-specific fluents:

- *list(a)*: a fluent meaning that the seller has listed property *a* (asset)
- *offer(a)*: a fluent meaning that the buyer has made an offer on property asset *a*
- *sign(a)*: a fluent meaning that both the buyer and seller have signed the purchase agreement for property *a*
- *pay(m)*: a fluent meaning that the buyer has paid the amount *m* agreed upon
- *transfer(a)*: a fluent meaning that the Notary has transferred the property *a*

Commitments:

- *intentToSell (a, m)*: an abbreviation for *BC(S, B, list(a)),* meaning that the seller is willing to sell a certain property *a* to the market by listing it for price *m*.
- *intentToBuy (a, m)*: an abbreviation for *CC(B, S, offer(a), list(a))*, meaning that the buyer is intended to make an offer on a property, once there are listings available that meet his/her criteria.
- *promiseToBuy(a, m)*: an abbreviation for *CC(B, S, sign(a), pay(m))*, meaning that the buyer is willing to pay for the property after signing (if the listing matches his/her budget).
- *promiseToSell(a, m)*: an abbreviation for *BC(S, B, sign(a))*, meaning that the seller is willing to sign the purchase agreement the buyer signs the purchase agreement that includes the requirements of his/her listing
- *deal(a,m)*: an abbreviation for *promiseToBuy(a, m)* $\wedge$ *promiseToSell(b, m),* whereafter the seller receives a full payment from the buyer.
- *exchange*(*a, m*): an abbreviation for *BC(S, B, transfer(a)),* meaning that change of ownership from seller to buyer is established.

The following protocol run can be derived from our simplified running example. Since this process is regulated and almost immutable, there is only once protocol run possible. Here e1..e9 are event calculus events that correspond to interface messages of the smart contract.

e1 List property (listProperty(a))
e2 Make an offer (makeOffer(a, m))
e3 Receive the offer (receiveOffer(a, m))
e4 Accept the offer (acceptOffer(a, m))
e5 Create the purchase agreement (createPurchaseAgreement(a, m))
e6 Sign the purchase agreement contract (SignPurchaseAgreement(a, m))
e7 Make the required payment (makePayment(m))
e8 Receive the payment (receivePayment(m))
e9 Exchange of ownership (exchangeProperty(a))

We focus on events that evolve our defined commitments (in bold) Hereby, *Create(e, x, c)* establishes commitment *c*. The create operation can *only* be performed by the debtor of the commitment, in this case the seller of the property. 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 [6]. For the (bold) events, we first specify the effects in terms of fluents:

A1. Initiates(listProperty(a), list(a), t)
A2. Initiates(makeOffer(a, m), offer(a), t)
A3. Initiates(signPurchaseAgreement(a, m), sign(a), t)
A4. Initiates(exchangeProperty(a), transfer(a))

Then we stipulate the effects in terms of commitments:

A5. Create(listProperty(a), S, intentToSell(a, m))
A6. Create(makeOffer(a, m), S, intentToBuy(a, m))
A7. Create(signPurchaseAgreement(a,m), S, promiseToSell(a, m))
A8. Create(signPurchaseAgreement(a,m), S, promiseToBuy(a, m))
A9. Create(receivePayment(m), S, exchange(a))

Base commitmens are initiated when they are created:

R1 *Initiates(e, C(x, y, q), t) ← Happens(e, t) ∧ Create(e, BC(x,y,q), t)*

The behavior of conditional commitments is slightly more complex. In line with [3], we implement Colombetti's definitions of conditional commitments, where a new (base) commitment is created when *p* is satisfied. This new commitment is created to satisfy *q*. So when a CC holds and an event happens that satisfies *p*, the CC is terminated and a C is initiated, as by the following axioms:

R2 *Initiates(e, C(x, y, q), t) ← HoldsAt(CC(x, y, p, q), t) ∧ Happens(e, t) ∧ Initiates(e, p, t)*

R3 *Terminates(e, CC(x, y, p, q), t) ← HoldsAt(CC(x, y, p, q), t) ∧ Happens(e, t) ∧ Initiates(e, p, t)*

Discharge(e, x, c) is an operation that resolves the commitment *c*. The discharge operation can only be performed by the debtor of the commitment to confirm that the commitment has been fulfilled. Discharging a commitment terminates that commitment.

When the seller lists the property on the market (e1), clause A5 is applied, initiating the intentToSell base commitment to a prospective buyer using rule R1:

Initiates(listPropertyt(a, m), intentToSell(a, m), t2) ← Happens(listProperty(a, m), t1) ∧ Create(listProperty(a, m), S, intentToSell(*a, m*))

Once a buyer makes a suitable offer on the property (e2), Axioms A6 at *t2* creates the *intentToBuy* conditional commitment. We do not terminate the *intentToBuy* commitment here, as the purchase agreement is not signed yet (until then we intent to buy), hence we delay termination until *p* is satisfied at *t5.* Together with rule R1:

Initiates(makeOffer(a, m), intentToBuy(a, m), t2) ← Happens(receiveOffer(a, m), t2) ∧ Create(receiveOffer(a, m), S, intentToBuy(*a, m*))

At *e6*, the buyer signs the purchase agreement first to satisfy *p* at *t5*, thereby creating the *promiseToSell* base commitment to satisfy *q* at *t4*. Once the seller has signed, the *promiseToSell* is discharged and the *intentToBuy* and *intentToSell* commitments are terminated accordingly at *t6* and *t7*. With rules R2 and R3:

Initiates(signPurchaseAgreement(a), promiseToSell(a, m), t4) ← HoldsAt(promiseToBuy(a,m), t3) ∧ Happens(signPurchaseAgreement(a), t3) ∧ Initiates(signPurchaseAgreement(a)), promiseToBuy(a,m), t3)

Terminates(intentToBuy(a, m)), t5) ← HoldsAt(promiseToBuy(a)), t5) ∧ Happens(signPurchaseAgreement(a), t5) ∧ Initiates(signPurchaseAgreement(a), sign(a), t5)

Terminates(promiseToSell(a, m)), t6) ← HoldsAt(promiseToSell(a,m)), t6) ∧ Happens(signPurchaseAgreement(a), t6) ∧ Initiates(signPurchaseAgreement(a), sign(m), t6)

Discharge(signPurchaseAgreement(a), S, intentToSell(a, m)) ←
HoldsAt(promiseToSell(a, m), t7) ∧ Happens(signPurchaseAgreement(a), t7) ∧
Initiates(signPurchaseAgreement(a), sign(a), t7)

Discharge(signPurchaseAgreement(a), S, promiseToSell(a, m)) ←
HoldsAt(promiseToSell(a, m), t8) ∧ Happens(signPurchaseAgreement(a), t8) ∧
Initiates(signPurchaseAgreement(a), sign(a), t8)

When the seller receives payment for transfer (*e9*), the *deal* commitment is initialized as *p* is satisfied at *t9*. As a result, the *exchange* base commitment is created to satisfy *q* at *t10*. Once ownership has been transferred from S to B at *t11*, the exchange base commitment is discharged by the seller.

Initiates(transferProperty(a), exchange(a), t10) ← HoldsAt(deal(a,m), t9) ∧
Happens(receivePayment(m), t9) ∧ Initiates(receivePayment(m), pay(m), t9)

Terminates(deal(a, m)), t9) ∧ HoldsAt(deal(a,m)), t9) ∧ Happens(receivePayment(m),
9t) ∧ Initiates(receivePayment(m), pay(m), t9)

Discharge(transferProperty(a), S, exchange(a), t11) ← HoldsAt(exchange(a), t11) ∧
Happens(transferProperty(a), t11) ∧ Initiates(transferProperty(a), transfer(a), t11)

CommitRuleML for CBSCs

Reaction rules are concerned with the invocation of actions in response to (complex) events and actionable situations. Reaction RuleML is the quasi-standard for representing reaction rules since its introduction in 2006, as it is regarded as a user-friendly XML-serialized sublanguage of RuleML. It acts as an interchange format for reactive rules and rule-based event-processing languages [7]. Reaction rules using Reaction RuleML typically implement forward-chaining operational semantics for Condition-Action rules where changing conditions trigger update actions, like IF/THEN/ELSE (derivative reasoning), IF/DO (production rules), ON/DO (trigger rules) or ON/IF/DO (Event-Condition-Action or ECA) [8]. In order to determine the best conceptual and semantic model for CBSC, various meta-model requirements have been formulated; (1) Since we consider commitments as social economic events that contain other events (e.g. to pay), the rules themselves should be event oriented. (2) The events should be detectable, which allows IF conditions to be specific for detected events only. (3) The meta-model should allow multiple event definitions to be part of the same rule procedure, in order to process a contract goal as a bundle of reciprocal commitments. (4) It should be possible to pre-define variables that apply to the entire smart contract. Finally, (5) In adherence to [1], the algorithms for logic approaches like have been made efficient and 'cheap' as measured within the environment where they are deployed and according to its economic rules.

Reaction rules include distributed Complex Event Process-ing (CEP), Knowledge Representation (KR) calculi, as well as Event- Condition-Action (ECA) rules, Production (CA) rules, and Trigger (EA) rules [8]. Since the event calculus reasons about effects of automated actions (on the blockchain) and the commitments that result from them, we leverage KR Reaction RuleML for CBSC. KR ReactionRuleML focus on inferences that are made from happened or planned events/actions (increment or decrement), they thereby describe inferences about the effects of these events/actions on changeable properties of the world (fluents/states) [7].

CommitRuleML has been designed in the same vein as RuleML extensions like PolicyRuleML and LegalRuleML. It converts the event calculus axioms into the various

properties that have been defined on top of KR Reaction RuleML. For example, if we would moderately extend KR Reaction RuleML, the metadata would semantically look like:

<!ELEMENT Head (Commitment)>
<!ELEMENT Interface (on, if, do)>
<!ELEMENT on | do (Event)>
<!ELEMENT if (Condition)>
<!ELEMENT do (Event)>
<!ELEMENT BC| CC | PC (Event)>

The CommitRuleML syntax for a base commitment as presented in our running example implements the metadata for shorter notation:

Initiates(receiveOffer(a, m), intentToBuy(a, m), t2) ← Happens(receiveOffer(a, m), t2) ∧ Create(receiveOffer(a, m), S, intentToBuy(*a, m*)) can be written in CommitRuleML as

```
<on><Happens><Event key="receiveOffer"/><at><Time>t2</Time></at></Happens></on>

<do>
    <Initiate>
       <Event key="receiveOffer"/>
              <BC key=”intentToBuy”>
                     <at><Time>t2</Time></at>
              </BC>
       </Event>
     </Initiate>
</do>
```

Conditional commitments can be written as follows:

Initiates(transferProperty(a), exchange(a), t10) ← HoldsAt(deal(a,m), t9) ∧ Happens(receivePayment(m), t9) ∧ Initiates(receivePayment(m), pay(m), t9)

```
<on>
   <And>
       <Happens><Event
key="receivePayment"/><at><Time>t9</Time></at></Event></Happens>
       <Initiate><Situation><fluent key=”pay”></fluent></Situation</Initiate>
   </And>
</on>
<if><Holds><Event key="deal"/><at><Time>t9</Time></at></Event></Holds></if>
<do>
    <Initiate>
       <Event key="transferProperty"/>
              <BC key=”exchange”><at><Time>t10</Time></at></BC>
       </Event>
     </Initiate>
</do>
```

Conclusion

This short paper aimed to present a way to provide an introduction into the formal syntax and semantics for CommitRuleML. The formalization process follows a similar pattern as defined by [8], leveraging Event Calculus and KR ReactionRuleML in order to define commitments for CBSCs. We have illustrated throughout a brief running example how base-, conditional and persistent commitments can be defined in an eCommerce transaction, and how these transactions evolve and fulfill over time. Future research could extend the running example by more complex use cases that include persistent commitments, partial fulfillments (e.g. payments) and discourse resolution. We think that the combination of CommitRuleML and blockchain's autonomous execution capability with immutable and agreed upon logic, has the potential to be a very powerful instrument to manage contracts in the future.

References

1. Idelberger, F., Governatori, G., Riveret, R., Sartor, G.: *Evaluation of Logic-Based Smart Contracts for Blockchain Systems*, Evaluation of Logic-Based Smart Contracts for Blockchain Systems, Rule Technologies. Research, Tools, and Applications: 10th International Symposium, RuleML 2016, Stony Brook, NY, USA, July 6-9, 2016.
2. McCarthy, J., Hayes, P.: *Some Philosophical problems with the standpoint of artificial intelligence*, Readings in Artificial Intelligence, 1981, Pages 431-450
3. Yolum, P., Singh, M.: *Reasoning About Commitments in the Event Calculus*: An Approach for Specifying and Executing Protocols, Annals of Mathematics and Artificial Intelligence, September 2004, Volume 42, Issue 1–3, pp 227–253
4. Kowalski, R., Sergot, M.: *A Logic-based Calculus of Events*, Foundations of Knowledge Base Management (pp.23-55)
5. De Kruijff, J., Weigand, H.: *Ontologies for Commitment Based Smart Contracts*, On the Move to Meaningful Internet Systems. OTM 2017 Conferences: Confederated International Conferences: CoopIS, C&TC, and ODBASE 2017, Rhodes, Greece, October 23-27, 2017, Proceedings, Part II (pp.383-398)
6. Paschke, Athan, T., Boley, H.: *Glossary of Reaction RuleML 1.02*
7. Paschke, A., Boley, H., Athan, T.: *The RuleML Perspective on Deliberation-Reaction Standards, Ontolog Rules Reasoning LP*: Series Session 5, 9 January 2014
8. Paschke, A., Boley, H., Zhao, Z., Teymourian, K.: *Reaction RuleML 1.0: Standardized Semantic Reaction Rules*, Complex Reactivity with Preferences in Rule-Based Agents (pp.100-119).