A REC-Based Commitment Tracking Tool

[System Demonstration]

Federico Chesani
University of Bologna
federico.chesani@unibo.it

Paola Mello
University of Bologna
paola.mello@unibo.it

Marco Montali
University of Bologna
marco.montali@unibo.it

Paolo Torroni
University of Bologna
paolo.torroni@unibo.it

1. INTRODUCTION

Social commitments are commitments made from an agent to another agent to bring about a certain property. In broad terms, a social commitment represents the commitment that an agent, called *debtor*, has towards another agent, called *creditor*, to bring about some property or state of affairs, which is the *subject* of the commitment.

Commitments are a well-known concept in Multi-Agent Systems (MAS) research [2, 6]. Representing the commitments that the agents have to one another and specifying constraints on their interactions in terms of commitments provides a principled basis for agent interactions [8].

Central to the whole approach is the idea of manipulation of commitments: their creation, discharge, delegation, assignment, cancellation, and release, since commitments are stateful objects that change in time as events occur. Time and events are, therefore, essential elements.

In our previous research [7] we introduced an abstract architecture for the specification and reasoning about social commitments. It supports the $\mathcal{CML}$ Commitment Modeling Language, and it is based on a reactive version of the Event Calculus [5] called $\mathcal{REC}$ [3]. In this demo, we show a concrete instantiation of the framework and the functioning of its prototypical version implemented in Java+Prolog.

2. CML ARCHITECTURE

The architecture that supports $\mathcal{CML}$, represented in Figure 1, consists of 4 layers: a **user application** top layer, one for **commitment modeling** below, a further **temporal representation and reasoning** layer, and a **reasoning and verification** one at the bottom.

On the top layer, the user can define agent social interaction rules, or contracts, using commitments. Such definitions are based on a language provided by the layer below. The commitment modeling language is implemented using a temporal representation and reasoning framework, which is in turn built on top of a more general reasoning and verification framework, which lies at the bottom layer. It is important to rely on a formal framework that accommodates various forms of verification, because in this way commitments can be operationalized and the user can formally analyze commitment-based contracts, reason on the state of commitments, plan for actions needed to reach states of fulfillment, and track the evolution of commitments at runtime.

We gave a concrete instantiation of such an architecture. We use it to implement the first $\mathcal{CML}$ prototype.

At the bottom layer, there are a number of Prolog+CLP modules which implement the SCIFF family of proof-procedures and provide the SCIFF language to the layer above [1]. The SCIFF framework is based on abductive logic programming and it consists of a declarative specification language and a family of proof-procedures for reasoning from SCIFF specifications. Some kinds of reasoning are: deduction, hypothetical reasoning, static verification of properties, compliance checking and run-time monitoring. In general, SCIFF comes in hand for a number of useful tasks in the context of agent interaction. A high-level description of SCIFF and of its usage is given in [8], also in relation with commitments. The CLP solvers integrated in SCIFF can work with discrete and dense domains, depending on the application needs, and they are particularly useful for reasoning along the temporal dimension.

On top of the SCIFF layer there is the SCIFF implementation of the $\mathcal{EC}$, $\mathcal{REC}$. There are several implementations for $\mathcal{EC}$. One of them which uses ideas taken from $\mathcal{CEC}$ and thus enables runtime verification, is called the Reactive Event Calculus ($\mathcal{REC}$) and it is implemented in SCIFF. The Event Calculus is a powerful formalism for temporal representation and reasoning introduced by Kowalski and Sergot [5]. $\mathcal{REC}$ is its reactive axiomatization, described in [3], and it is implemented as a SCIFF program. This layer provides to the layer above the $\mathcal{REC}$ language, which consists of domain-dependent axioms.

In the third layer, the constructs that define the Commitment Modeling Language ($\mathcal{CML}$), i.e., the notation proposed above, are written by way of $\mathcal{REC}$ theories. Thus this layer will provide the language to write a $\mathcal{CML}$ Program ($CProgram$) to the top layer.

The top layer consists of user and domain-dependent knowledge encoded into a $CProgram$.

WOODSTOCK '97 El Paso, Texas USA

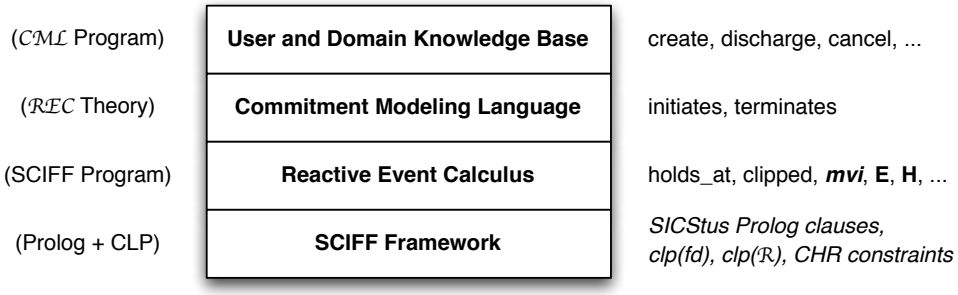

Figure 1: Social commitment framework architecture

A $CProgram$ is made of rules. A rule in the form

$$CRuleHead \leftarrow CRuleBody \tag{1}$$

is used to define effects of events on the state of commitments. More specifically, the user can use such rules to define for instance which events create, discharge, or break which commitments, in the style of [9]. The body defines the context, i.e., the conditions that must hold in order for an event to have an effect on the state of a commitment. If no context is defined, the rule is said to be a fact.

Atoms in the body of rules may be external predicates, not defined in the commitment specification program (this should be allowed because of the modularity principle), or they can be fluents modeling the state of commitments. Such a state can be associated with existentially quantified temporal variables ($holds\_e$ notation) or with universally quantified intervals ($holds\_u$ notation).

3. EXAMPLE

The example that we use in the demo is the following.

Let us consider a car rental company r and a customer c. The former guarantees that its cars will not break down for at least two days, promising an immediate replacement if one does.

This situation can be represented by the $CProgram$ that follows, where by $\mathbf{H}(break\_down(T))$ we denote an event occurred ("**H**appened") at time T:

$$\begin{aligned} &create(rent\_a\_car(T_c, T_e), \mathsf{C}(r, c, [T_c, T_c + 2]great\_car), \_). \\ &create(break\_down, \mathsf{C}(r, c, [T_r]replace\_car), T_b) \leftarrow \\ &\qquad T_r \leq T_b + 1 \wedge \\ &\qquad holds\_at(viol(\mathsf{C}(r, c, [T_s, T_e]great\_car), T_b), T_b). \end{aligned} \tag{2}$$

The first rule says that renting a car creates a commitment $\mathsf{C}(r, c, [T_c, T_c+2]great\_car)$ that the car will not break down in the first two days. The commitment is created the time the car is rent, thus it is left unbound ($\_$).

The second rule says that if the car breaks down in the first two days, by which a commitment $\mathsf{C}(r, c, [T_c, T_e]great\_car)$ is violated, then another commitment is created about a replacement car to be sent before 1 day. Here we explicitly denote by T_b the time the car breaks down, because the commitment is created only if the violation holds at the same time.

The $\mathcal{EC}$ predicate $holds\_at(F, T)$ means that a given fluent F holds at a time T [5]. Additionally, our language provides predicates $holds\_e(F, T_1, T_2)$/$holds\_u(F, T_1, T_2)$ to indicate that a given fluent F holds at some point/at all points inside the time interval $[T_1, T_2]$.

In this demo, we use the $\mathcal{CML}$ Tool based on the $CProgram$ above. In particular, as events occur (car is rented, car breaks down, time passes, car is replaced, etc) commitments are created, discharged, violated, etc. $\mathcal{CML}$ Tool shows the state of commitments in time, as events occur.

We will show several situations. Time granularity is one day.

Situation 1 At day 2, c rents a car from r for the period that goes from day 17 to day 20. This is represented by an event $\mathbf{H}(rent\_a\_car(17, 20), T)$. Such an event initiates a fluent representing a commitment, in which r is the debtor, and c the creditor, and the subject $great\_car$ indicates that the car will not break down. At day 18 the car breaks down. At day 19 the car is replaced. As events occur, $\mathcal{CML}$ Tool shows that a commitment $\mathsf{C}(r, c, [T_c, T_c + 2]great\_car)$ is created on day 2 and then violated on day 18, and then that a commitment $\mathsf{C}(r, c, [T_r]replace\_car)$ is created on day 18 and fulfilled on day 19.

Situation 2 As a variation of Situation 1, the replacement car is sent out on day 20. $\mathcal{CML}$ Tool shows a double violation.

Situation 3 As a variation of Situation 1, the car breaks down on day 13, and is replaced on day 16. All happens before the rental period starts. $\mathsf{C}(r, c, [T_c, T_c + 2]great\_car)$ is not affected. $\mathcal{CML}$ Tool shows fulfillment after day 18, and it shows how $great\_car$, which is initially true, changes value as the car breaks down (becames false) and gets replaced (becomes true again).

Situation 3 As a variation of Situation 1, the car breaks down on day 20. $\mathcal{CML}$ Tool shows that no commitment is violated.

4. TOOL DESCRIPTION

Let us now briefly describe the system implementation from a technological and engineering perspective.

4.1 System requirements

The problem addressed is commitment tracking. The system requirements are

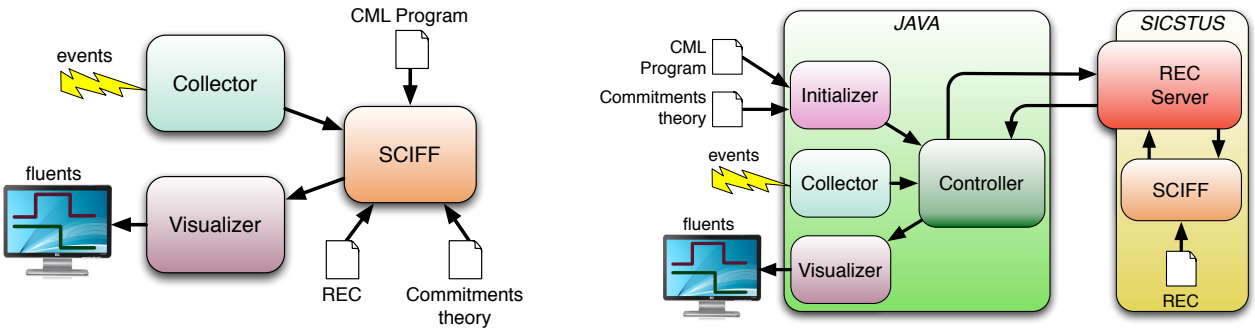

Figure 2: Logical system architecture

- a formalism to describe events that are relevant for commitment tracking and manipulation
- a formalism to describe the theory of commitments
- an operational framework that receives events at runtime, and based on a commitment theory and background knowledge outputs the state of commitments along the temporal dimension
- there are two execution modes: runtime (the user inputs events one by one), or simulated runtime (events are taken from a logged event narrative and the system reasons from events one by one as if they were actually produced at runtime)
- the end user must be able to use the tool by inputting events (or by selecting a logged event narrative), and by defining *CPrograms*, but without having to describe the underlying theories ($\mathcal{REC}$, SCIFF, ...)
- output should be easily readable (e.g., graphics)

4.2 Design and implementation

Reasoning is done by the SCIFF engine, which is implemented in SICStus Prolog and CHR. The user interface is achieved via a Java module which receives events and *CPrograms* in inputs, queries the SCIFF engine, and returns a pictorial representations of commitment states. These states are captured by fluents (commitment x is active, commitment x is violated, property y holds, ...).

The logical system architecture is depicted in Figure 2. In particular, Figure 2(a) shows the reasoning engine (SCIFF) data flow. The SCIFF reasoner needs $\mathcal{REC}$ theory, *CProgram* and the theory of commitments. It takes events from an event collector module and produces the maximum validity intervals of fluents. Figure 2(b) describes the interplay between Java and SICStus. A Java thread takes events in input and interfaces with the SCIFF reasoner. Such a reasoner is a SICStus Prolog server, which provides 4 methods (predicates):init, startREC, fetchevent, and endREC, and outputs the set of all fluents in the form of an XML structure. Such an XML structure is sent back to the Java interface, which produces and updates the graphic representation of fluents (see the screenshot in Figure 3).

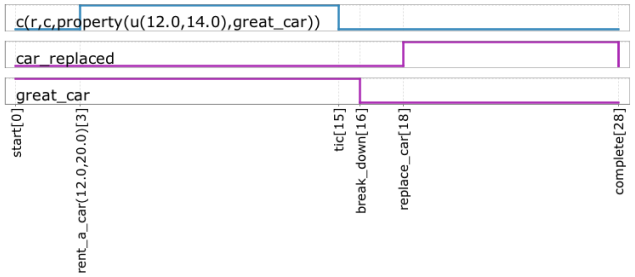

Figure 3: Output of the $\mathcal{REC}$-based tool

4.3 Perspectives

We have tested the system using examples from literature [7],[4]. The $\mathcal{CML}$ system is still at a prototypical state. It has not been used to address "real" case studies (yet). However, the need of such a tool has emerged from an analysis of real-world scenarios. We hope to find the resources we need to engineer the $\mathcal{CML}$ system and propose it to an audience of practitioners. A possibility we are investigating is to wrap $\mathcal{REC}$ into a plugin for widespread tools such as ProM,[1] as we already did with another SCIFF-based tool called SCIFFchecker.

5. REFERENCES

[1] M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Verifiable agent interaction in abductive logic programming: the SCIFF framework. *ACM Transactions on Computational Logic*, 9(4):1–43, 2008.

[2] C. Castelfranchi. Commitments: From individual intentions to groups and organizations. In V. R. Lesser and L. Gasser, editors, *Proceedings of the First International Conference on Multiagent Systems*, pages 41–48. The MIT Press, 1995.

[3] F. Chesani, P. Mello, M. Montali, and P. Torroni. Commitment tracking via the reactive event calculus. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*. AAAI Press, 2009.

[4] F. Chesani, P. Mello, M. Montali, and P. Torroni. Verification of choreographies during execution using

[1] `http://prom.sourceforge.net/`

the reactive event calculus. In *Web Services and Formal Methods, 5th International Workshop, WS-FM 2008, Milan, Italy, September 4-5, 2008, Revised Selected Papers*, volume 5387 of *Lecture Notes in Computer Science*, pages 55–72, Berlin Heidelberg, 2009. Springer Verlag.

[5] R. A. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1):67–95, 1986.

[6] M. P. Singh. An ontology for commitments in multiagent systems: Toward a unification of normative concepts. *Artificial Intelligence and Law*, 7:97–113, 1999.

[7] P. Torroni, F. Chesani, P. Mello, and M. Montali. Social commitments in time: Satisfied or compensated. In *Proceedings of the 7th International Workshop on Declarative Agent Languages and Technologies (DALT)*. `http://www.di.unito.it/~baldoni/DALT-2009/`, 2009.

[8] P. Torroni, P. Yolum, M. P. Singh, M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, and P. Mello. Modelling interactions via commitments and expectations. In V. Dignum, editor, *Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models*, pages 263–284, Hershey, Pennsylvania, Mar. 2009. IGI Global.

[9] P. Yolum and M. Singh. Flexible protocol specification and execution: applying event calculus planning using commitments. In *Proc. 1st AAMAS*, pages 527–534. ACM Press, 2002.