ACE: a Flexible Environment for Complex Event Processing in Logical Agents

Stefania Costantini

University of L'Aquila, Italy, Imperial College London, UK

EMAS 2015@AAMAS, Istanbul

Events Recognition and Evaluation: from ECAs to CEP

- Event processing: traditionally based upon Event-Condition-Action (ECA) rules, of the form

  IF Event then Reaction

  where sometimes reaction is limited to a sequence of actions, sometimes entails forms of reasoning and various kinds of constraints.
- Most (virtually all) agent-oriented languages provide ECA rules of some form.
- In many cases: the interpretation of sets of simple events is not univocal or straightforward: different hypotheses should be generated and evaluated.

Complex Event Processing

- Complex Event processing (CEP) has emerged as a relevant new field of software engineering and computer science.
- In fact, many practical applications have the need to actively monitor vast quantities of event data to make automated decisions and take time-critical actions.
- CEP is particularly important in software agents.

Complex Events Recognition and Evaluation

Needs memory of previously-identified events and their correlation.

- Important approaches to CEP based upon Event-Calculus, as time and time intervals are important.
- Recent approaches (ETALIS, oclingo) provide forms of stream reasoning, and time intervals.
- DyKnow: a knowledge processing middleware framework providing software support for creating streams from inputs gathered from distributed sources representing high-level events concerning aspects of the past, current, and future state of a system.

Complex Event Processing: Our Approach

- Our approach is concerned mainly (but not only) with logical agents, i.e., agents whose syntax and semantics is rooted in Computational Logic. CEP in Logical Languages: DALI, GOAL, ETALIS, KGP, METATEM, 3APL, etc.
- We propose a novel conceptual view of complex events and Event-Processing Agents (EPAs)
- We observe in fact that often the occurrence of a complex event cannot just be detected from deterministic incremental aggregation of simple events.
- We introduce special modules, specifying the possible interpretations of the occurrence of certain events in a certain time frame, and actions to be undertaken according to the chosen interpretation.

Our Proposal: Agent Computational Environment (ACE)

ACE can be considered as a generalization of such work, in that ACE is:

- agent-oriented;
- aimed at managing heterogeneity in the definition/description of knowledge sources, that moreover can interact among themselves and with external sources;
- aimed at providing a uniform semantics of single components and of the overall system;
- aimed at allowing for verification of properties.

ACE: Augmented Agents

- Agents equipped with special modules for event interpretation and, symmetrically, action generation. Such modules:
  - are triggered by a set of simple events occurring in some combination and/or within a certain time interval;
  - generate possible (alternative) interpretations of such simple events as complex events according to conditions, agent's previous experience, context, constraints, etc.
- An agent can choose the 'preferred' interpretation based upon plausibity/probabiity/learning mechanisms, etc.

Event-Action Modules

Definition

We let an *Event-Action module* be defined as $M = (tr_M; L_M{}^{\mathcal{P}}; kb_M; br_M;)$ where tr_M is an event expression which triggers the module evaluation, $L_{M_i}^{\mathcal{P}}$ is a preferential logic, kb_M is a knowledge base in this logic, and br_M is a set of bridge rules as defined for MCSs.

An mMCS (Managed Multi-Context System) $M = (C_1, \ldots, C_n)$ is a heterogeneous collection of contexts $C_i = (L_i; kb_i; br_i)$ where L_i is a logic, kb_i is a knowledge base in L_i and br_i a set of bridge rules of the form:

$$o(s) \leftarrow (c_1 : p_1), \ldots, (c_j : p_j), not\,(c_{j+1} : p_{j+1}), \ldots, not\,(c_m : p_m).$$

where the new item of information s is added to kb_i as $o(s)$ after some kind on context-specific Mr. Senthil Annnamalai elaboration, or “management”.

Event-Action Modules Activation

Definition

An Event-Action module M is *active* w.r.t. sequence Π of sets of events (or simply "active" if leaving Π implicit) iff $ev^{\mathcal{E}}(tr_M, \Pi) = true$, i.e., if Π enables the module evaluation w.r.t. evaluation function $ev^{\mathcal{E}}$.

A Sample DALI Event-Action Module

The *TRIGGER* Section

An Event-Action Module will be activated whenever the *triggering* events occur within a specified time interval, and according to specific conditions.

$$EVENT-ACTION-MODULE\ Diagnosis$$
$$TRIGGER$$
$$(high\_temperatureP\ AND\ intense\_coughP) : [2days]$$

The module will be re-evaluated every time that a new occurrence of the triggering events should arrive.

A Sample DALI Event-Action Module

The *COMPLEX_EVENTS* Sections

From given symptoms, one can either a suspect flu or suspect pneumonia, or both. These are the complex events this module is able to cope with.

```
COMPLEX_EVENTS
suspect_flu OR suspect_preumonia
```

Sample DALI Event-Action Module

Conditions

Each of the listed complex events can be inferred, though according to specific conditions. The whole agent's belief base (including the history, and namely all past events that have been recorded) is implicitly included in the definition of an Event-Action Module.

$$
\begin{array}{l}
\textit{suspect\_flu} \text{ :- } \textit{high\_temperatureP}. \\
\textit{suspect\_pneumonia} \text{ :- } \textit{high\_temperatureP} : [\textit{4days}], \\
\qquad\qquad\qquad\qquad\qquad \textit{intense\_coughP}. \\
\textit{suspect\_pneumonia} \text{ :- } \ \%\ \textbf{Bridge Rule} \\
\qquad\quad \textit{diagnosis}(\textit{clinical\_history}, \textit{suspect\_pneumonia}) : \\
\qquad\quad \textit{diag\_knowledge\_base}. \ \%\ \textbf{External Context}
\end{array}
$$

A Sample DALI Event-Action Module
PREFERENCES Section

(Complex) preferences can be expressed about outcome plausibility.

PREFERENCES
suspect_flu :- *patient_is_healty*.
suspect_pneumonia :- *patient_is_at_risk*.

A Sample DALI Event-Action Module

MANDATORY Section

Some conclusions, however, are mandatory in certain conditions.

$$
\begin{array}{l}
\textit{MANDATORY} \\
\textit{suspect\_preumonia} :\!- \\
\qquad\qquad \textit{high\_temperatureP} : [\textit{4days}], \\
\qquad\qquad \textit{suspect\_fluP}, \\
\qquad\qquad \textit{assume\_antibioticP} : [\textit{2days}].
\end{array}
$$

A Sample DALI Event-Action Module

ACTIONS Section

Suitable actions to be undertaken according to the conclusions drawn can be specified.

```
ACTIONS
suspect_flu :> stay_in_bedA.

suspect_flu,
high_temperatureP : [4days],
not suspect_pneumonia       :>  assume_antibioticA.

suspect_preumonia :>  assume_antibioticA,
                      consult_lung_doctorA.
```

ACE: Formal Definition

Definition

An Agent Computational Environment (ACE) $\mathcal{A}$ is a tuple

$$\langle A, M_1, \ldots, M_r, C_1, \ldots, C_s \rangle$$

where, for $r, s \geq 0$, A is the “basic agent” (main agent program), the M_is are Event-Action modules and the C_is are contexts in the sense of MCSs. We put the following restrictions on bridge rule bodies: (i) both contexts and modules can be mentioned in bodies of bridge rules of A; (ii) both contexts and basic agent A can be mentioned in bodies of bridge rules of the M_is; (iii) only contexts can be mentioned in bodies of bridge rules of the C_is.

ACE: Agent-Module-Contexts Interaction

- *A* is an agent program specified in any logical agent-oriented language; bridge rules can be used as follows:
  - the basic agent can query via bridge rules every component;
  - Event-Action modules can query contexts, but also the basic agent;
  - contexts can only query other contexts.

ACE: Semantics (informal)

Semantics of ACE extends that of mMCSs

- In both mMCSs and in an ACEs a data state $S = (S_1, \ldots, S_h)$ includes a set of knowledge bases, one for each component of the system, where each S_i is derived according to the underlying logic, and in compliance with bridge-rules application.
- Desirable data states (Equilibria) are those where each S_i is acceptable according to each component's policy of knowledge management and taking bridge rules application into account.

ACE: Semantics Cont'd (informal)

Differences w.r.t. mMCSs

- Bridge rule application in mMCS is unconditional (every applicable rule is applied).
- In ACEs, a bridge rule is applicable only if each Event-Action module which is queried is also active.
- In ACEs, equilibria are defined w.r.t. event sequences.
- Given sequence of sets of events the corresponding ACE-Evolution is the sequence of equilibria.
- ACE-evolutions: defined in view of verification.

Event-Action Modules for Generating Complex Actions

Devising which actions and agent should perform can be highly context-dependent, and can be subjected to various kinds of uncertainty.

To avoid brittleness, an agent should in our opinion be able to flexibly choose what to do in specific circumstances, and to dynamically adapt to changes of context/role/situation.

Generating Complex Actions

Triggering Events/Actions

Actions performed either by the agent itself or by other agents are considered to be symmetric to each other.

$EVENT-ACTION-MODULE\ Encounter$
$TRIGGER\ meet\_friend(A, F),$

$COMPLEX\_EVENTS\_ACTIONS$
$smile(A, F) OR\ (wave(A, F)\ XOR\ shake\_hands(A, F))$

Generating Complex Actions

In the following (simplicistic) definition, agent either follows social conventions (if any) or repeats what the other does.

$$\begin{array}{l} \mathit{MANDATORY} \\ \mathit{shake\_handsA}(A, F) \text{ :- } \mathit{formal\_situation}(A, F). \\ \\ \mathit{ACTIONS} \\ \mathit{smileP}(A, F) :> \mathit{smileA}(F, A). \\ \mathit{waveP}(A, F) :> \mathit{waveA}(F, A). \\ \\ \mathit{shake\_handsP}(A, F) :> \mathit{shake\_handsA}(F, A). \end{array}$$

Generating Complex Actions

Action Preconditions

$PRECONDITIONS$
$smileA(A, F)$:- $not\ angry(A, F), not\ bad\_temper(A).$
$waveA(A, F)$:- $not\ angry(A, F).$

$shake\_handsA(A, F)$:-
  $good\_friends(A, F),$
  $not\ angry(A, F), not\ in\_a\_hurry(A), not\ in\_a\_hurry(F).$

Generating Complex Actions

Anomalies Detection

What if the other agent behaves differently from expected?

```
ANOMALY
anomaly1(meet_friend(A, F)) :-
        smileP(A, F), not smileA(F, A).
anomaly2(meet_friend(A, F)) :-
        waveP(A, F), not waveA(F, A).
anomaly3(meet_friend(A, F)) :-
        shake_handsP(A, F), not shake_handsA(F, A).

ANOMALY_MANAGEMENT_ACTIONS
...
```

Procedural Semantics and Implementation

The DALI solution

Each DALI Event-Action Module can be translated in a fully automated way into an Answer Set Programming (ASP) module, as DALI is equipped with an ASP plugin.

ASP is a well-established fully logical logic programming paradigm, where a program may have a number (none, one or several) answer sets expressing possible solutions.

Sections *ACTIONS*, *ANOMALY_MANAGEMENT_ACTIONS* and *PRECONDITIONS* do not need translation, as they are just sets of plain and reactive rules.

Future Directions

- Other event aggregation, recognition and generation patterns.
- Continuous learning for adaptation to evolving context/environment.
- ’Deep’ learning: Event-Action Modules defined in a tentative or embryonic form, then learnt via a suitable training phase and refined according to agent’s subsequent significant experiences.
- Integration with probabilities, deontic logics, theory of emotions . . .
- A-priori verification and run-time assurance via Temporal Logic.
- Full efficient implementation.

The End!

Thank You for your Attention:-)
Questions?