SAGE: A Logical Agent-Based Environment Monitoring and Control System

Krysia Broda[1]*, Keith Clark[1], Rob Miller[2], and Alessandra Russo[1]

[1] Department of Computing, Imperial College London
`www.doc.ic.ac.uk/{~kb/,~klc/,~ar3/}`
`{k.broda,k.clark,a.russo}@imperial.ac.uk`
[2] Department of Information Studies, University College London
`www.ucl.ac.uk/infostudies/rob-miller/` `rsm@ucl.ac.uk`

Abstract. We propose SAGE, an agent-based environment monitoring and control system based on computation logic. SAGE uses forward chaining deductive inference to map low level sensor data to high level events, multi-agent abductive reasoning to provide possible explanations for these events, and teleo-reactive programming to react to these explanations, e.g. to gather extra information to check abduced hypotheses. The system is embedded in a publish/subscribe architecture.

Key words: Environmental Control, Logic, Event Calculus, Logic Programming, Abduction, Multi-Agent Reasoning, Teleo-Reactive Programs

1 Introduction

SAGE (Sense, Abduce, Gather, Execute) is an agent based environment monitoring and control system we are building based on computational logic. Its key components are: (1) the use of the event calculus [7] and forward chaining deduction to map low-level time-stamped sensor reading events into inferred higher-level composite events using specialist agents for each type of sensor, (2) the use of multi-agent abductive reasoning [6] to co-operatively infer unobserved events or actions as possible explanations of the composite events, (3) the use of agents executing persistent teleo-reactive [9] control procedures that automatically respond to possible explanations, to gather auxiliary information about the state of the environment or to execute goal directed and robust control responses, (4) the use of a formal logical ontology to specify application dependent event terms, facts and action descriptions, and (5) the use of publish/subscribe servers [12] as a communication infrastructure enabling easy integration of new components and agents that have only to conform to the application ontology regarding the notifications they publish and the subscriptions they lodge.

The key advantage of using computational logic is that it allows high level declarative representation of an application domain. It greatly facilitates extensibility and re-engineering for different domains.

Throughout this paper we will provide further commentary on the features listed above with reference to the following example scenario. Ann, Bob and Carl live in a sheltered housing complex which employs SAGE. At 9:30:00am and 9:30:09am respectively, adjacent sensors s34 and s35 detect movement down the main corridor of the complex. The building agent infers that someone is

* Authors are listed in alphabetical order, not in order of their relative contribution.

moving down the corridor and consults with the intelligent agents of its residents to either confirm or eliminate them as a possible explanation. Ann's agent confirms the likelihood that it is her, because she has an dental appointment at 10am that requires her to leave the building, so the building agent positions the lift appropriately. Later, the building agent infers a similar but faster movement down the same corridor, but this time Ann's and Bob's agents rule themselves out as possibilites, because Ann is at the dentist, and Bob's recent ankle sprain prevents him from moving that quickly. Carl's agent has learned that if he forgets to take his medication he tends to wander around the building, and so prompts the nurse to check his medication pack. The nurse confirms that Carl has forgotten his medication and so Carl's agent is therefore unable to confirm his likely whereabouts. So the building agent seeks another source of confirmation by asking the security agent to locate Carl, which it attempts by displaying a query message on the (human) security guard's computer screen. The system simultaneously considers the possibility of an intruder, and seeks extra confirmation of this by asking the perimeter camera agents to swivell round searching for signs of forced entry. Camera c92 locates a broken fence, and there is no previous log of this information. Meanwhile the security guard ascertains that Carl is elsewhere and responds to the computer query accordingly. The building agent now has sufficient evidence to support the explanation that an intruder is in the building, and reacts accordingly, locking doors, alerting the staff, etc.

Notation: using Prolog convention, variables start with uppercase and constants with lowercase. Variables are universally quantified with maximum scope.

2 Agent Communication Via Publish and Subscribe

To integrate the various components of SAGE we use a publish/subscribe server called Pedro [12] which matches subscriptions with notifications using Prolog unification technology. Additionally, Pedro supports peer-to-peer address based communication, as commonly used in agent applications and assumed in the FIPA agent communications language [11]. Asynchronous messaging systems that support publish/subscribe message routing [2] have been found useful for developing open distributed applications [8], and particularly so for complex event processing [5] and open multi-agent applications such as ours.

Pedro messages are strings representing Prolog terms. For example, a subscription is a string of the form "`subscribe(T,Q,R)`" where `T` is a message template (a Prolog term, usually with variables), `Q` is an associated Prolog query using variables that appear in `T`, and `R` is an integer which can be used by the subscriber as a subscription identifier. Pedro automatically forwards any notification it receives to all processes that have a current subscription `S` that *covers* the notification, preceded by its identifier `R`. A subscription *covers* a notification `N` iff the Prolog query `T=N,Q` succeeds. Here, `=` is term unification, a generalisation of pattern matching in which both `T` and `N` can contain variables.

As an example of the use of Pedro, a motion detection sensor `s34` might send the notification "`motionDetected(s34,time(9:30:00))`" at time 9:30. This is covered by the subscription:

```
subscribe(motionDetected(S,_), (member(S,[s34,s35]), 1)
```

which might have been lodged by the agent monitoring a corridor with two sensors s34 and s35. If so, Pedro forwards the notification to this agent, preceded by the subscription identifier 1. If the same agent now receives a similar notifications from the sensor s35 mentioned in its subscription, with time value 9:30:09 it might infer and then post the notification:

```
movement(hall1,vel(south,6),during(9:30:00,9:30:09))
```

to Pedro to be picked up by any (possibly unknown) agent interested in this information. Such an interested agent will have lodged a subscription such as:

```
subscribe(movement(Pl,Vel(Dir,Sp)),(3=<Sp,Sp=<8), ..)
```

The range restriction on Sp is because the agent is only interested in movement of a walking or running person, not, say, an electric cart.

The key advantage of using Pedro for an application which involves both event processing and control responses is that all that has to be decided is the ontology for event notifications and control messages. The system is then open. As monitoring agents are added they subscribe for the event notifications of interest to them, which they can update at any time. These agents then attempt to exert control over the monitored system by issuing action requests. No component needs to know the identities of other components, or even what other components there are.

3 Interpreting Sensor Data Via Forward Chaining

The corridor monitoring agent subscribing to the sensor readings in our example might use an implication such as the following to deduce (by forward chaining) the movement notification that it posts:

```
[motionDetected(S1,T1) ∧ motionDetected(S2,T2)  ∧
 coLocated(S1,S2,Loc,Dir,Dis) ∧ speed(Dis,T1,T2,Sp)]
                      → movement(Loc,Vel(Dir,Sp),during(T1,T3))
```

where the `coLocated` and `speed` conditions are part of the agent's background knowledge. This is a (much simplified) example of an event calculus "counts as" rule, defining a complex event in terms of simpler ones. The event calculus additionally allows us to infer persisting properties that are initiated and terminated by events, e.g. the event of moving the lift from floor 2 to floor 1 initiates the property that it is at floor 1 and terminates it being at floor 2.

4 Explanation Generation Via Distributed Abduction

As mentioned above, SAGE uses *abduction* to generate possible explanations of (directly or indirectly) observed changes or events in its environment. In the scenario of Section 1, for example, various explanations for the detected corridor movement are abduced by different agents. Formally, abduction is the process of finding a set of sentences Δ that can be (consistently) added to a theory T so that $T \wedge \Delta \models G$, for a given "goal" G. Δ cannot contain arbitrary sentences but must be composed only of "abducible" sentences (usually literals), the set of which is domain-specific. In the present context Δ is an explanation of an event or change in circumstance G (e.g. corridor movement) deduced from sensor data. The environmental knowledge T is spread among different agents (the building agent, the residents' agents, etc.) as logic programs with integrity constraints, and so we make use of the distributed logic program abductive procedure DARE [6]. Distributed abduction involves the cooperation of different logical agents in constructing explanations Δ which draw upon their combined knowledge and respect their combined consistency requirements. DARE has the advantage, crucial to SAGE, of being an *open* proof procedure: agents can join or leave the cooperative abductive process during its execution without affecting the correctness of the outcome. This openess is facilitated by the use of Pedro (see Section 2) for inter-agent communication during the abduction.

As a (simplified) example, in our scenario the building agent may have a rule:

```
movement(Loc,vel(Dir,Sp),during(Begin,End)) ←
        3 ≤ S ∧ S ≤ 8 ∧ at(Person,Loc,walking,during(Begin,End))
```

which can be invoked by matching its head with the details of a particular detected movement `movement(hall1,...)` that needs explaining. Anne's agent is able to help satisfy the conditions of this rule by adding `at(anne,hall1,...)` to the current set Δ, since her set of abducibles contains literals of this form, and it is consistent with current beliefs and conjectures. Later when Anne is at the dentist her agent will be unable to abduce a similar explanation because of the integrity constraint `¬[at(X,L1,...) ∧ at(X,L2,...) ∧ L1≠L2]`.

5 Teleo-reactive and Information Gathering Procedures

For agent actions we use Nilsson's TR (Teleo-Reactive) procedures [9] for robot control embedded in a multi-threaded agent architecture [13]. TR procedures have the form:

```
p(X1,..,Xk){c1 -> a1.    c2 -> a2.   ...    cn -> an}.
```

where X1,..,Xk are parameters and the body is an ordered sequence of condition-action rules. The conditions can access the current store of observed and inferred (i.e. deduced or abduced) events and any other beliefs about the state of the environment and inhabitants. The event and belief stores are continuously and asynchronously updated as a procedure executes.

The rules' conditions are constantly re-evaluated to find the first rule with a true condition to fire. Once a rule is fired its action is persistently executed until *another* rule is fired. A special `exit` action terminates a procedure. The actions can be information gathering actions that indirectly result in new sensor events (e.g. video capture images) being added to the event store, which may result in another rule being fired. The following is a TR procedure that might be invoked to pan a surveillance camera to look for a sign of entry through a perimeter fence. `FT` is the time when the panning should terminate if the analysis of the camera images has not detected some sign of entry, such as a broken fence:

```
lookForSignEntry(FT){
      broken_fence or gate_open or current_time > T -> exit.
      true  -> pan_camera. }.
```

6 Conclusions and Related Work

The key features of SAGE are that it provides a flexible, distributed, open and component-based approach to environmental monitoring and control, and that its computational processes reflect natural, multi-stage, collaborative human reasoning: when events are detected it forms and tests hypotheses about these before reacting appropriately. Its multi-agent and multi-threaded architecture allows it to form and act upon different hypotheses concurrently. Space limitations prevent us from commentating on all the features of SAGE hinted at in our example scenario, such as agent learning capability and human intervention supplementing its reasoning processes. Work on SAGE is currently at the specification stage (hence this short paper format), although some of its key components already have implementations. Our next stage will be to provide a full implementation and to test this with a simulated example environment, using the event calculus to "run" different scenarios and measure the likely success of our approach.

Space limitations prevent a detailed comparison of related work, but [1] is another ambient intelligent system that uses a publish/subscribe architecture to integrate a disparate set of components, [4] uses the event calculus for a distributed agent environment, [3, 10] are examples of logic based agent architectures for ambient intelligent systems.

References

1. M. Anastasopoulos et al., *Towards a Reference Middleware Architecture for Ambient Intelligence Systems*, Building Software for Pervasive Computing, OOPSLA05, 2005.
2. R. Baldoni, M. Contenti and A. Virgillito. *The Evolution of Publish/Subscribe Communication Systems*, Future Directions of Distributed Computing, Springer Verlag LNCS Vol. 2584, 2003.
3. A. Bikakis and A. Grigoriou, *Distributed Defeasible Contextual Reasoning in Ambient Computing*, AMI'08, 2008.
4. S. Bromuri and K. Stathis, *Distributed Agent Environment in the Ambient Event Calculus*, DEBS'09, 2009.
5. D. Luckham, *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*, Addison Wesley Professional, May 2002.
6. J. Ma, A. Russo, K. Broda and K. Clark, *DARE: a system for distributed abductive reasoning*, Autonomous Agent Multi-Agent Systems, Vol 16, pp. 271-297, 2008.
7. R. Miller and M. Shanahan, *Some Alternative Formulations of the Event Calculus*, Lecture Notes in Articial Intelligence, vol. 2408, pp. 452–490, 2002.
8. G. Muhl, L. Fiege and P. Pietzuch, *Distributed Event-Based Systems*, Springer, 2006.
9. N. Nilsson, *Teleo-Reactive Programs for Agent Control*, Journal of Artificial Intelligence Research, 1:139-158, 1994.
10. K. Stathis and F. Toni, *Ambient Intelligence Using KGP Agents*, EUSAI 2004, LNCS 3295, 2004.
11. Foundation for Intelligent Physical Agents, *Fipa Communicative Act Library Specification*, www.fipa.orgspecs/fipa00008/XC00008H.html, 2002.
12. P. J. Robinson and K. L. Clark, *Pedro: A Publish/Subscribe Server Using Prolog Technology*, submitted to Software Practice and Experience, 2009.
13. K. L. Clark, *AgentMT(TR): a Multi-threaded Agent Architecture Using Teleo-Reactive Plans*, in preparation.