Reconciling Autonomy with Narratives in the Event Calculus

L. Chen, K. Bechkoum, G. Clapworthy

Department of Computer and Information Sciences
De Montfort University
Hammerwood Gate, Kents Hill
Milton Keynes, MK7 6HP, U.K.
Email: {lchen, kbechkoum, gc}@dmu.ac.uk

Abstract

Developing believable and realistic characters for interactive, computer-based forms of entertainment is a hard work. To make them perform specific tasks or take initiatives given a narrative is even more challenging. In this paper we introduce a novel agent design approach that reconciles autonomy with instructability and narrative in one agent architecture. The approach is based on a highly developed logical theory of action and a powerful high-level behaviour specification language (BSL) that is developed from the underlying logical formalism, i.e. the event calculus. Using BSL, agents' behaviours can be specified and controlled more naturally and intuitively, more succinctly and at a much higher level of abstraction than would otherwise be possible. We also briefly discuss the implementation issues relevant to this approach.

1. Introduction

Building animated human-like agents (also known as autonomous actors, autonomous creatures, and synthetic character) is an active research area and much progress has been made towards character's geometric and low-level behavioural modelling. However, at present most characters used for interactive, computer-based forms of entertainment are self-interested autonomous agents. Simply situating animated agents in simulated virtual environments and letting them behave autonomously is not what we expect for computer-based interactive entertainment and the emerging Internet-based VR applications such as virtual learning, conferencing. We hope that users can control agents' behaviour at different level of abstraction. For example, users may prefer to delegate tasks to an agent by issuing high-level commands or let agents take initiative by giving the agent a narrative. At the extreme end, users may have the capability of controlling the agent at motivational level. They can have the agent's belief, desire and intention instantiated with their motivation, preference and other personality traits, and the agent will behave exactly like the users themselves.

There are many research works on animated agent (Badler et al. 1993)(Terzopoulos 1994) (Reynolds 1987). Most of them are built based on a behavioural model with a reactive agent architecture. Though robust and efficiency, the disadvantage of the approach is that the behaviour controllers are hardwired into code. Blumberg and Perlin (Blumberg et al. 1995) (Perlin and Goldberg 1996) have begun to address such concerns by introducing mechanisms that give the animator greater control over behaviour. While we share similar motivations, our research takes a different route. We place the emphasis on investigating important higher-level cognitive abilities, such as knowledge representation, reasoning, and planning, which are the domain of AI.

Funge and Lespérance (Funge et al. 1999) (Lespérance et al. 1994) present a cognitive model for character animation based on the Situation Calculus. Both of us try to define and implement cognitive model for animated agents; however, our work is different from Funge's in that we adopt a different logical formalism for reasoning action and time, i. e. Event Calculus (EC) (Shanahan and Witkowski 2000). Although the two formalism share the basic ontology of atomic actions and fluents, event calculus has the advantage of representing actual actions, in particular, the actions with duration that is the most common actions in character animation (Kowalski and Sadri 1994). Another salient feature of the EC is its ability to assimilate a narrative, i.e. the description of a course of events such as plot or story, adjust the effects of actions and the time-line of the narrative as it become more and more precise in an additive only fashion. These innate features of the formalism ground our high-level control specification language on a solid theoretical foundation and make it more suitable for modelling and controlling the behaviour of animated agents.

To help build cognitive models and facilitate users' control over the agent behaviour, a high-level behaviour specification language (BSL) is developed. This high-level language provides an intuitive way to specify agents' knowledge about their domain in terms of actions, their preconditions and their effects. We can also endow agents with a certain amount of "common sense" within their domain. Therefore, when we give an agent instructions or narratives we can leave out tiresome details from the commands. The missing details are automatically filled in at run-time by the character's reasoning engine that decides what must be done to achieve the specified goal.

In this paper we introduce and develop cognitive modelling methodology for high-level agent control. Cognitive models go beyond current implementation of behavioural models and reactive agent architecture in that they govern what an agent knows, how that knowledge is acquired, and how it can be used to plan actions. Comparing to traditional AI style planning. The distinguishing feature of our approach is the development

and exploitation of the high-level behaviour specification language for domain specification, reasoning engine design and controllers' programming. This language forms an important middle ground between regular logic programming (as represented by Prolog) and traditional imperative programming (as typified by C++). Moreover, with this approach, the level of control, the power of the reasoning engine and the run-time efficiency can be tuned to achieve optimal performance in terms of the application requirements. For example, if you give more domain knowledge and increase the power of the reasoning engine, then you can control the agent at a higher level. On contrary, the system can also run with less domain knowledge, a weak reasoning engine but detailed, lower level of control.

The remainder of the paper is organised as follows. Section 2 introduces the theoretical basics of the event calculus and the planning mechanism in the EC. Section 3 presents the high-level behaviour specification language (BSL) and the methodology for programming high-level controllers. In section 4, we briefly describe an agent architecture that facilitates the logical approach to reconcile autonomy with instructability. Section 5 provides a design example for virtual agents. Section 6 presents conclusions and suggestions for future work.

2. Theoretical Foundations

Our approach to integrating autonomy and narratives into one agent architecture is to use a logical formalism to model virtual worlds from the animated agent's point of view. The formalism for reasoning about action is based on many-sorted first-order predicate calculus augmented with circumscription (Shanahan 1997). The advantage of the event calculus over other logical formalisms such as the situation calculus is that the time flow is represented independently from the notion of an action and the actions are embedded in this independent structure using an explicit notion of an action occurrence. This feature makes it ideal for reasoning narratives, that is, reasoning about actions that actually occur at various times and reasoning about the properties that hold or do not hold at different times as a consequence of such occurrences. Below we introduce the basics of the event calculus and the semantics of compound actions that underpin the high-level behaviour specification language.

2.1 The Basics of the Event Calculus

The event calculus is a logical programming formalism for representing and reasoning about events and their effects. The ontology of the event calculus comprises fluents, events (or actions) and time points. Events are the fundamental instrument that brings about changes to the world. Any property of the world that can change over time is known as a fluent. A *fluent* is a function of the time point. The event calculus uses predicates to specify actions and their effects. For example, predicate **Initiates(**α, β, τ**)** means that the fluent β starts to hold after event α at time τ; predicate **Happens(**$\alpha, \tau 1, \tau 2$**)** means that event α starts at time $\tau 1$ and ends at time $\tau 2$.

Initiates(α, β, τ) [Fluent β starts to hold after action α at time τ]
Terminates(α, β, τ) [Fluent β ceases to hold after action α at time τ]
Releases(α, β, τ) [Fluent β is not subject to inertia after α at time τ]
InitiallyP(β) [Fluent β hold from time 0]
InitiallyN (β) [Fluent β does not hold from time 0]
Happens(α, τ1, τ2) [Action α starts at time τ1 and ends at time τ2]
τ1<τ2 [Time point τ1 is before time point τ2]
HoldAt(β, τ) [Fluent β holds at time τ]
Clipped(τ1, β, τ2) [Fluent β is terminated between times τ1 and τ2]
Declipped(τ1, β, τ2) [Fluent β is initiated between times τ1 and τ2]
Cancels(α1, α2, β) [The occurrence of α1 cancels the effect of a simultaneous occurrence of α2 on fluent β]
Cancelled(α, β, τ1, τ2) [Other concurrent event occurs from time τ1 to time τ2 that cancels the effect of action α on fluent β]
Trajectory(β1, τ, β2, δ) [If fluent β1 is initiated at time τ then fluent β2 becomes true at time τ +δ]

Table 1. The language of the event Calculus

The predicates of the event calculus are listed in Table 1. Based on the causal relation of the predicates in the event calculus we can derive a set of axioms. The conjunction of the collection of the axioms is denoted the EC. Each axiom states how and when a fact (or a proposition) will hold. Here are two examples of these axioms.

$$\text{HoldAt}(f,t3) \leftarrow \text{Happens}(a,t1,t2) \wedge \text{Initiates}(a,f,t1) \wedge \neg \text{Cancelled}(a,f,t1,t2) \wedge t2<t3 \wedge \neg \text{Clipped}(t1,f,t3)$$

$$\text{Clipped}(t1,f,t4) \leftrightarrow \exists a,t2,t3[\text{Happens}(a,t2,t3) \wedge t1<t3 \wedge t2<t4 \wedge [\text{Terminates}(a,f,t2) \vee \text{Releases}(a,f,t2)] \wedge \neg \text{Cancelled}(a,f,t2,t3)]$$

The latest version of the extended event calculus, i.e. the predicates and axioms, is formally described in (Shanahan 1997). The frame problem of the formalism is overcome through circumscription. This is similar to the closed world assumption. The ramification problem is the frame problem for actions with indirect effects. We introduce state constraints to sort out this problem.

2.2 Planning in the Event Calculus

Planning in the event calculus is an abductive reasoning process through resolution theorem prover. The logical definition of EC-planning is as follows.

Given a conjunction Σ of Initiates, Terminates and Realeases formulae describing the effects of actions (a domain description), a conjunction Δ_0 of InitiallyP and InitiallyN describing the initial situation, a finite conjunction ψ of state constraints describing actions' indirect effects, and a conjunction Ω of uniqueness-of-names axioms for actions and fluents.

We represent a goal Γ as a finite conjunction of formulae of the form,

HoldAt(β, t) or $\neg$ HoldAt(β, t)

Where β is a ground fluent and t is a ground time point.

Suppose we use a narrative of events to denote a finite conjunction of Happens formulae and temporal ordering among them.

Then a plan for Γ is a narrative Δ such that,

$$\text{CIRC}[\Sigma; \text{Initiates, Terminates, Releases}] \wedge \text{CIRC}[\Delta_0 \wedge \Delta; \text{Happens}] \wedge \psi \wedge \text{EC} \wedge \Omega \models \Gamma$$

Where, CIRC means after circumscription.

This gives rise to the formal specification of planning definition. The circumscriptive solution to the frame problem adopted here accommodates the inclusion of state constraints, so long as they are conjoined to the theory outside the scope of any circumscription.

3. Agent Control Mechanism

3.1 Compound Actions and the BSL

The previous sections outline the basics of the event calculus and the related approach for representing and reasoning about simple actions. It fails to address the problem of expressing and reasoning about compound actions such as "**If** action α is successful **then** do action β **else** do action δ", "**While** there are friends **do** greetings **endwhile".** Our solution to this problem is to introduce some additional extra-logical symbols such as **|, ?, while, if,** etc., which act as abbreviations for logical expressions in the event calculus. Then we represent compound actions as a combination of primitive actions and other compound actions in terms of these extra-logical symbols. For convenience we denote compound actions as procedures. They are actually macros of primitive actions and other compound actions. During execution, compound actions expand into genuine formulae of the event calculus.

Based on the EC formalism, the introduced extra-logical symbols and the compound action semantics, we develop the high-level behaviour specification language (BSL). Table 2 lists the definition of the main extra-logical symbols and the corresponding BSL syntax in square brackets. The BSL can be used to specify an agent's behaviour by manipulating events with the assistance of those logical symbols. A program in BSL can be viewed as a top-level procedure that consists of combinations of sequence and/or parallel of procedures and primitive actions. All the procedures and actions within the program are linked together through the extra-logical symbols. Program execution is accomplished via macro-expansion into sentences of the event calculus. Although the syntax of BSL is similar to a conventional programming language, BSL is a strict superset in terms of functionality.

The user can give agents a single command or a narrative in terms of available behaviour controllers. In particular, a behaviour controller can be nondeterministic so that one instruction can covers multiple possibilities. Given a high-level instruction, an agent can decide to fill in the necessary missing details by itself according to its domain knowledge and behave autonomously.

(Sequence) $\alpha;\beta$ means do action α, followed by action β. [**take**<ACTION α> **then** <ACTION β>]
(Nondeterministic choice of actions) $\alpha\|\beta$ means do action α, or action β. **choose** <ACTION α> **or** <ACTION β>]
(Test) *p*? means true if formula *p* holds, otherwise false. [**test** <EXPRESSIONS *p*]
(Nondeterministic choice of arguments) $(\pi x)\ \alpha(x)$ it means pick some argument *x* and perform the action $\alpha(x)$. **pick** (<EXPRESSION x>) <ACTION>]
(Concurrency) $\alpha \mid\mid \beta$ means actions α and β occurs concurrently. [**take**<ACTION α> **and** <ACTION β>]
(Non-deterministic iteration) α^* means do α zero or more times. [**iterate** <ACTION α>]
(Iteration) while *p* do α means do α while *p* is true. [**while** (<EXPRESSION *p*>) < ACTION α>]
(Conditionals) if *p* then α else β means do α if *p* is true, otherwise do β. [**if** (<EXPRESSION *p*>) **then** <ACTION α> **else** <ACTION β>]
(Concurrency with different priorities) $\alpha >> \beta$ means that α has higher priority than β, and may only be executed when α is done or blocked. [**take**<ACTION α> **then and** <ACTION β>]
$< \dot{X} : \varphi \rightarrow \alpha >$ means if formula φ is true for binding of the variable vectors $\dot{X}$, then interrupt current action and execute the action body α [**if** (<EXPRESSION φ ($\dot{X}$)>) **interrupt** <ACTION α>]
(Procedures) **Proc** *P*(λ1, λ2, ...λn) α **endproc** declares a compound action [**void** P(<ARGLIST λ1, λ2, ...λn >) <ACTION α>]

Table 2. The Extra-logical Symbols and the SBL Syntax

3.2 Control Programming

EC-based agent programming is to use the logic formalism as a representation medium and theorem proving as a means of computation. A high-level control program in such system is a set of logical formulae describing actions including compound actions, their effects and their temporal relation. To program an agent, we need first specify the domain in which the agent operates, so the agent know what the domain is about, how to acquire knowledge and how to use them to plan actions to achieve goals. In the event calculus a complete domain description usually comprises two sets of Initiates, Terminates and

Releases formulae describing the effects of the agent's primitive low-level actions and the compound high-level actions respectively, a set of Happens formulae defining high-level compound actions and a set of declarations specifying state constraints.

To operate in a complex, dynamic and unpredictable virtual world, agents should be able to perform perception and update their knowledge when necessary. We introduce a generic epistemic fluent Knows as discussed in [14] to represent and reason about knowledge and the knowledge producing actions. This fluent has exactly the same status as other fluents and can be formalised in the same way as other fluent's formalisation. For example, the formula HoldAt(Know(ϕ), τ) represents that the formula ϕ follows from the agent's knowledge or agent's beliefs at time τ.

The agent executes a sense-plan-act cycle. Initially the agent has an empty plan and is presented with a goal Γ in the form of HoldAt formulae. Using resolution against formulae in domain specification, the planning process identifies a high-level compound action α that will achieve Γ. The planning process then decomposes α using resolution against formulae that define high-level compound actions. If the decomposition produces executable, primitive actions, then they are be added to the plan. If the decomposition yield further sub-goals (HoldAt formulae) or further compound sub-actions (Happens formulae), then we will repeat the above process to identify the compound actions corresponding to the sub-goals or further decompose the compound sub-actions. This process, i.e. the loop of goal, compound action and decomposition, continues until a complete plan is arrived. If there is no corresponding high-level compound action available to a given goal in domain specification, the agent will use the formulae describing primitive actions to make plan from first principles as discussed in section 2.2.

This algorithm interweaves sensing, planning and acting together. Sensor events or states serve as pre-conditions for subsequent actions or conditionals for choice control. They can also be used as conditions for terminating current actions. Planning is a resolution-based abductive theorem proving process working on a collection of event calculus formulae. The most salient feature is the exploitation of hierarchical planning via compound actions. This facilitates planning in progression order, which promotes the rapid generation of a first executable action. If a sensing produces an unexpected event or state that fails to satisfy the required conditions for subsequent actions, then the agent will be required to re-plans from scratch.

The control programs, written in the event calculus and the high-level behaviour specification language (BSL), have both a declarative meaning, as a collection of sentences of logic, and a procedural meaning, given by the control algorithm through compound action.

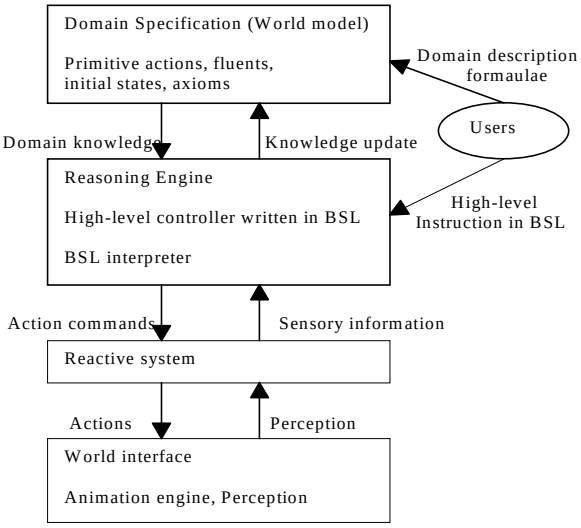

Figure 1. A Simplified Agent Architecture

4. Agent Architecture

We propose a layered hierarchical agent architecture for cognitive modelling as shown in Figure 1 (Chen et al. 2000). The core framework of the architecture consists of three levels of control module. At the lowest level, the animation engine provides believable human appearance and realistic motion. At the middle level, the reactive layer is responsible for generating reactive responses for unexpected events. At the highest level, the reasoning engine implements the agent's brain and is responsible for motor, perception and low-level action control. The reasoning engine provides agents with a cognitive model that enable them to reason about their world based on acquired knowledge, thus enabling them to interpret high-level instructions from human users. One feature of the layered agent control model is that it allows the agent to be controlled at different levels of abstraction. Thus in the reasoning engine we need only consider high-level actions (or behaviour) such as "greeting friends", "having drink". The reactive system and the animation engine translate these commands down to detailed degree of freedom The user supplies the primitive actions and their effects on the world, the specification of the initial state, and the domain-dependent controllers. The top-level controllers programmed in BSL drive the agent behaviour. The BSL interpreter executes these programs using the axioms and domain knowledge to generate a complete plan for a given goal. However, the high-level controller does not drive the platform directly, but through a low-level reactive system that perform some reactive actions to deal with unexpected events such as collision avoidance. This frees the high-level controller from having to respond to exceptional conditions in real time. The reactive system can also act as a fail-safer should the reasoning system temporarily fall through. In the event that the agent can not decide upon an intelligent course of action in a reasonable amount of time, the reactive system will invoke an emergent behaviour just to keep the agent ongoing.

5. An Example

We use the BSL and the event calculus to program an agent to accomplish various activities in a virtual campus. The agent can behave autonomously and/or take initiatives through user's instruction. The top-level controller is showed below.

```
proc agentBehaviourController
kact_initializeAgent;
while fl_agentActivatedStates = active do
< fl_userInstruction(p) →
comact_handleInstruction>
>>
< fl_generatedGoals(g) →
comact_handleGoal(g)>
>>
< fl_agentStates = Idle →
comact_selfAmusing>
endwhile
endProc
```

Where the variables with prefix kact_ are knowledge producing actions; the variables with prefix_comact_ are compound actions and the variable with prefix fl_ mean fluents.

The top priority interrupt takes care of handling users' instructions. This ensures the agent always responds to a user interaction prior to reacting to the agent's internal intention. Each command can be a simple primitive action or compound action representing a narrative. The agent will follow the narratives given by the user to accomplish delegated tasks. At the second level of priority, the interrupt is responsible for agent's internal reaction to environmental changes. Without user's interaction, the agent will make decisions according to its environmental changes and take actions autonomously. At the lowest level, if the agent has nothing to do for a while, it will play a self-amusing procedure to reduce boring or to attract attention.

The compound actions contained in the controller are developed in terms of the control programming methodology discussed in section 3.2. The following procedure is one of the high level controller for handling user's instructions.

```
Proc handleInstruction(p)
if (!fl_goalAchieved(fl_currCommand) ∧
(commandPrio(p) < commandPrio(fl_currCommand)) then
comact_achieve(fl_currCommand)
>>
comact_achieve(p)
else
comact_achieve(p)
endif
endproc
```

6. Conclusion

This paper introduces a logical approach to high-level agent control, which uses logic as a medium of representation and theorem proving as a means of computation. By means of the developed behaviour specification language, this approach can reconcile autonomy with instructability and narratives in one agent architecture. We believe that this novel modelling methodology will greatly facilitate the animated agent design for interactive entertainment and virtual environment applications over the Internet.

Cognitive agent modelling is still at the early stage of research. Here we only outline the theoretical background and briefly discuss the implementation issues. The main purpose is to share our ideas with other researchers and get new inspiration from the feedback.

References:

Badler, N. I., Phillips, C. and Webber, B. L. 1993. Simulating Humans: Computer Graphics, Animation and Control. Oxford University Press. New York.

Blumberg, B. M. and Galyean, T. A. 1995. Multi-level direction of Autonomous Creatures for Real-time Virtual Environment. In Proceedings of SIGGRAPH95, 47-54.

Chen L., Bechkoum K and Clapworhty G. 2000. An Animated Human-like Interface Agent for Virtual Environments. In Proceedings of the 4th International Conference on Computer Graphics and Artificial Intelligence, 29-39, Limoges, France.

Funge J., Tu X., and Terzopoulos D. 1999. Cognitive Modeling: Knowledge, Reasoning and Planning for Intelligent Agents, SIGGRAPH 99, Los Angeles, CA.

Kowalski R. and F. Sadri. 1994. The situation calculus and event calculus compared. Proceedings of the international symposium on logic programming (ILPS'94), 539-553.

Lespérance Y., H. Levesque, F. Lin, D. Marcu, R. Reiter, and R. Scherl. 1994. A Logical Approach to High-Level Robot Programming - A Progress Report. In Proceedings of the 1994 AAAI Fall Symposium, 79-85, LA.

Perlin K. and A. Goldberg. 1996. IMPROV: A system for scripting interactive actors in virtual worlds. In *Proceedings of SIGGRAPH 96*, 205–216.

Reynolds, C. W. 1987. Flocks, Herds, and Schools: A Distributed Behavioural Model. In Proceedings of SIGGRAPH87, 27-31, Anaheim, CA.

Shanahan M. P. & M.Witkowski. 2000. High-Level Robot Control Through Logic, Proceedings ATAL 2000

Shanahan M. P. 1997. Solving the Frame Problem: A Mathematical Investigation of the Common Sense Law of Inertia, MIT Press

Terzopoulos D., Tu, X. and Grzeszczuk, R. 1994. Artificial Fishes: Autonomous Locomotion, Perception Behaviour and Learning in a Simulated Physical World. In Artificial Life Four, 327-351