Conscious Error Recovery and Interrupt Handling

Haythem O. Ismail and Stuart C. Shapiro
Department of Computer Science and Engineering
and Center for Cognitive Science
State University of New York at Buffalo
226 Bell Hall
Buffalo, NY 14260-2000
{hismail|shapiro}@cse.buffalo.edu

Abstract *We present a model for error recovery and interrupt handling by a reasoning, acting, and linguistically-competent cognitive agent. Faced with an emergency situation, the agent reasons about what it needs to do and what it is currently doing to decide what to do next. Its reasoning is based on general context-sensitive domain knowledge about the priority of acts. Such knowledge may be provided to the agent in natural language while it is acting, rather than being hardwired into the agent's knowledge base as is the case with most existing systems.*

Keywords: Acting, interrupt handling, reasoning about action, cognitive robotics.

1 Introduction

In the real world, any of the actions of a cognitive agent may fail to achieve its goals. A theory of acting agents should therefore be constructed with failure deeply in mind. The agent should be aware at each step of the outcome of its previous actions and should behave appropriately to recover from errors. In addition, such behavior should be the result of the agent's reasoning, not of hardwired reactive mechanisms. In particular, the agent should be able to reason and discuss its actions and failures.

Our theory is based on the GLAIR agent architecture [1, 2]. This is a layered architecture, the top layer of which is responsible for high level cognitive tasks such as reasoning and natural language understanding. This level is implemented using the SNePS knowledge representation and reasoning system [3, 4, 5]. We use "Cassie" as the name of our agent. Previous versions of Cassie have been discussed elsewhere [6, 7]. Those are actually embodied versions of the disembodied linguistically-competent cognitive agent of the SNePS system discussed in previous work [8, 9].

Cassie should be capable of using natural language to interact with other agents (possibly human operators). This means that SNePS representations of the contents of Cassie's memory ought to be linguistically-motivated. By that we mean two things. First, on the technical side, the representations should be designed so that they may be produced by a natural language understanding system, and may be given as input to a natural language generator. Second, at a deeper level, the syntax of the representations and the underlying ontology should reflect their natural language (in our case, English) counterparts. In particular, we admit into the SNePS ontology anything that we can think or talk about [3, 4]. For a general review of linguistically-motivated knowledge representation, see [10]. Among the things that Cassie should be capable of talking about are her own actions, her intentions, and the relative priorities of her different acts.

The paper is organized as follows. In Section 2 we review related work. In Section 3 we provide an in-depth analysis of the concept of interruption. Section 4 describes *cascades*—our model for pessimistic conscious sequential acting. A formal characterization of prioritized acting is presented in Section 5. Interrupt handling and our model for perception are described in Section 6. Finally, in Section 7, we present our conclusions.

2 Related Work

The action literature within symbolic artificial intelligence contains various, though essentially similar, proposals to deal with the problem of interrupt handling. The basic recurring theme is that interrupt handling involves the definition of priorities among acts (or goals). Reactive planners [11, 12, for instance] typically interleave planning and action execution; once the planner generates a primitive act, it starts to execute while planning is still going on. Interrupts in this setting may be handled by simply generating the appropriate reaction. This is feasible since the system never commits to a particular sequence of actions, only one act at a time.

Other systems, where plans are specified (or generated off-line) in some action specification language, need to provide appropriate means for handling interrupts. [13] presents precise semantics for an action language equipped with control structures for handling interrupts. For example, the expression "interrupt_for$(T1, T2)$" corresponds to the execution of task $T2$, interrupting it when necessary for $T1$. Essentially, this means that $T1$ has higher priority over $T2$. Given the formal semantics [13, p. 42], it is not clear how the system can represent priorities that may change over time. Such an issue, we believe, is crucial and, as it turns out, is overlooked by many of the existing systems.

Within the GOLOG family [14], interrupts are handled in CONGOLOG using special control structures for priorities and reactions [15]. In a CONGOLOG program, the expression "$(\sigma_1\rangle\rangle\sigma_2)$ denotes the concurrent execution of the actions σ_1 and σ_2 with σ_1 having higher priority than σ_2" [15, p. 1224]. Note that this is essentially Davis' "interrupt_for" control structure. Further, "$(\sigma_1\rangle\rangle\sigma_2)$" is an act, a step in a CONGOLOG program that the agent should execute in a certain manner as indicated by the semantics. Thus, once the agent starts performing "$(\sigma_1\rangle\rangle\sigma_2)$, it is not obvious how it may decide to "change its mind" regarding the priorities of σ_1 and σ_2, and, for example, interrupt σ_1 to perform σ_2.

Interrupt handling obviously involves the notion of priorities. The problem with approaches such as the above (where priorities are represented as actions in plans) is that they do not provide enough flexibility for the agent to reason about what to do next. We present a system where priority information is represented as context-dependent domain knowledge that may be communicated on-line in natural language. Interrupt handling is not represented by means of explicit control structures, but is built into the acting executive. Whenever the agent is about to act, it reasons about what it is about to do, and what it is currently doing, to decide what to do next.

3 The Concept of Interruption

To develop a theory of recovering from errors, one needs to be precise about what sort of thing an error is. An error, as far as this work is concerned, is a special kind of an interrupt; an event that causes the agent to stop what it is doing and handle an unexpected situation. An error is special in that the unexpected situation is the failure of one of the agent's acts. A general theory for interrupt handling would subsume one for error recovery. We, therefore, shall discuss general interrupt handling in this document, error recovery being a by-product of our theory.

An interrupt is an event that causes the agent to change its intentions and/or actions. It involves three main components: an event η, a reaction $\rho(\eta)$, and a *non-empty* set Π of on-going processes. For there to be an interrupt, the three components have to be present. For example, a robot may be carrying out a number of concurrent processes (Π) when its battery goes low (η) requiring it to move to the recharging station ($\rho(\eta)$). A number of points to note:

1. The reaction is a function of the event η. Thus, there can be no situation in which a reaction is present without a corresponding event.

2. The event η may be an instruction by another superior agent (possibly a human operator) to perform some action, which, in that case, would be $\rho(\eta)$.

3. The set Π is not empty. If the agent is not doing anything, then whatever happens is not an interrupt, just an event that may require some appropriate reaction.[1]

4. A valid reaction is the act of stopping one of the processes in Π.

For the sake of simplicity, let us assume for the moment that Π is a singleton containing only one act, A. A could be either *primitive* or *composite*. Primitive acts are ones that are performed by the agent "subconsciously"; it can perform them, but cannot reason about how it does. In particular, the agent cannot *explain* (in English, for instance) how it performs its primitive acts. Different control structures (sequential, conditional, iterative, etc.) form composite acts out of primitive acts. The agent is aware of the structure of its composite acts, and may discuss how it performs them (down to the primitive acts level). When η occurs (and is noticed by the agent), the agent could be performing either a primitive or a composite act. Corresponding to these two situations, there are two types of interrupts: one that happens while performing a composite act and another that happens in the midst of executing a primitive act. To make things more concrete, we can identify these two categories of interrupts as follows:

[1]Note that this subsumes errors. Nothing can qualify as an error if the agent is not doing anything.

1. The agent is executing a composite act A which reduces to the execution of some sequence of acts $\langle \alpha_1 \ldots \alpha_n \rangle$. The agent has just finished performing α_i, and is about to perform α_{i+1}, when η occurs. For example, the agent may be performing the sequence of acts ⟨`PickUp-Block`, `Goto-Table`, `Put-Down-Block`⟩ and have just picked up the block when it senses that its battery is low.

2. The agent is in the midst of executing a primitive act (which could be part of a composite act) when η occurs. For example, while performing the ⟨`PickUp-Block`, `Goto-Table`, `Put-Down-Block`⟩ sequence, the battery goes low while the agent is moving toward the table.

In the first case, the agent needs to merely change (or, more precisely, *revise*) its intentions regarding what to do next (go to the table or recharge the battery). In the second case, the agent may need to *stop* what it is doing to handle the interrupt. In any situation, there are two main requirement on any interrupt handling mechanism:

1. The agent should first perform the act with higher priority. If continuing to perform A is more important than performing $\rho(\eta)$, then this is what the agent should do. Note that priorities are context-sensitive, they change according to the current situation. For instance, if the agent is at the table, then putting the block down may have higher priority than recharging the battery. If, on the other hand, the agent is at the recharging station, then recharging the battery should have higher priority.

2. If it chooses to stop A and perform $\rho(\eta)$, the agent should resume A when $\rho(\eta)$ is complete. On the other hand, if it chooses to continue performing A, it should somehow *remember* to perform $\rho(\eta)$ when it is done.

4 Cascades

Consider an agent performing a sequence of acts $\langle \alpha_1 \ldots \alpha_n \rangle$. In order to be able to correctly interrupt such a composite act and then resume it, the agent needs to be aware of the progression of the act and to have some way of knowing what remains to be done. In other words, if the agent is performing step α_i, it should somehow remember that it still needs to perform the sequence $\langle \alpha_{i+1} \ldots \alpha_n \rangle$. In addition, an agent that anticipates failure, should start performing step α_{i+1} when and only when step α_i has been successfully completed. Our system readily provides these features through the sequencing control act `cascade` [16]. What `cascade` essentially does is initiate the first act in the sequence, wait for its goal to be achieved, and then *cascade* the rest of the sequence. This requires some method for *transforming* the belief that the goal has been achieved into an appropriate act. The `when-do` construct allows just that [17, 18, 19, 5]. Informally, if forward inference causes the propositions `when-do`(p, α) and p to be asserted, then the act α is performed and `when-do`(p, α) is disbelieved. That is, a `when-do`(p, α) proposition, represents the agent's belief that it should perform a certain act, α, as a reaction to its coming to believe some proposition p. Thus, if p is the proposition of event η having occurred, then α is $\rho(\eta)$.

Informally, to perform a cascade, `cascade`$(\alpha_1 \ldots \alpha_n)$, the agent first believes the proposition `when-do`$(\Gamma(\alpha_1)$, `cascade`$(\alpha_2 \ldots \alpha_n))$ and then performs α_1, where $\Gamma(\alpha)$ denotes the goal of the act α.[2] When the goal of α_1 is achieved, this will be asserted with forward inference, thereby causing the pending cascade, `cascade`$(\alpha_2 \ldots \alpha_n)$, to resume. By using `cascade`, we are essentially building a pessimistic agent that anticipates the failure of any of its acts. It executes a step in a sequence when and only when it comes to *know* (mainly based on perception and bodily feedback) that the goal of the previous step has been achieved. In addition, the believed `when-do` provides the bookkeeping required for the agent to remember what is yet to be done (the rest of the cascade). Thus, should an interrupt occur, the agent would have a way to resume what it was doing exactly at the point where it left off.

[2]This is a very simplified presentation of cascades. For space limitations, we do not give a full discussion. This has been done elsewhere [16].

5 Prioritized Acting

As pointed out in Section 3, appropriately handling interrupts requires the agent to reason about the priorities of its acts. Priorities define a partial order over the set of acts. Two acts that are not prioritized relative to each other are assumed to have the same priority.[3] Specification of priorities among acts may be explicitly represented in the knowledge base.

- *Holds*(*p-higher*$(\alpha_1,\ \alpha_2)$, t)

Where *p-higher*$(\alpha_1,\ \alpha_2)$ denotes a state in which the act α_1 has higher priority than the act α_2. As a whole, the above form means that, at time t, α_1 has priority over α_2. The important point here is that priorities are dependent on the over-all situation (see the discussion in Section 2). Priorities among general acts, including arbitrary cascades (that may be generated on the fly while carrying out some plan), are inductively defined as follows.

Definition 5.1 *Let α_1 and α_2 be two distinct acts. $\alpha_1 >_p \alpha_2$ (read, α_1 has higher priority over α_2) iff:*

1. $\alpha_1 =$ ***cascade***$(\alpha_i, \ldots, \alpha_{i+n})$ *and* $\alpha_i >_p \alpha_2$,
2. $\alpha_2 =$ ***cascade***$(\alpha_j, \ldots, \alpha_{j+m})$ *and* $\alpha_1 >_p \alpha_j$, *or*
3. Holds(p-higher$(\alpha_1,\ \alpha_2)$, ***NOW**) *is deducible.*[4]

[3]Note that two acts that make use of the same resources (cameras, wheels, etc.) should be appropriately prioritized.

[4]**NOW** is a meta-logical variable whose value, at any point, is the SNePS term denoting the current time. ***NOW** is a shorthand for the the value of **NOW**. See [20] for more details.

Accordingly, the relation $>_p$ holds between two cascades if it holds between their first elements. The base case of the induction is the explicit assertion of priorities (in terms of *p-higher*) among acts. The above does not say that α_1 should be completed before starting to perform α_2, it only defines the $>_p$ relation. What should be done when this relation holds between two acts is a different issue that we now turn to.

Definition 5.2 *Let $\mathcal{A}$ be a set of acts. Define the set $\mathcal{A}_\top$ as follows. An act $\alpha \in \mathcal{A}_\top$ iff:*

1. *$\alpha \in \mathcal{A}$, α is a cascade, and for every $\alpha' \in \mathcal{A}$, if $\alpha' \neq \alpha$ then* Holds*(*p-higher*(α, α'),* ***NOW***) is deducible;*
2. *$\alpha \in \mathcal{A}$, α is not a cascade, and there is no $\alpha' \in \mathcal{A}$ such that $\alpha' >_p \alpha$; or*
3. *$c =$ `cascade`$(\alpha, \ldots, \alpha_n) \in \mathcal{A}$ and $\alpha \in (\mathcal{A} \cup \{\alpha\})_\top$.*

Intuitively, $\mathcal{A}_\top$ is the set of acts in $\mathcal{A}$, or embedded within cascades in $\mathcal{A}$, that should be performed first, i.e, those with *top* priorities. A complementary set contains whatever remains.

Definition 5.3 *Let $\mathcal{A}$ be a set of acts. Define $\mathcal{A}_\perp = (\mathcal{A} - \mathcal{A}_\top) \cup \{$`cascade`$(\alpha_{2_i}, \ldots, \alpha_{n_i}) \mid$ `cascade`$(\alpha_{1_i}, \alpha_{2_i}, \ldots, \alpha_{n_i}) \in \mathcal{A}$ and $\alpha_{1_i} \in \mathcal{A}_\top\}$*

The above defines priorities among acts. To perform actions according to their priorities, we introduced the control act `p-do-all`. This is based on the `do-all` control act which initiates a set of acts in some arbitrary order.[5] Let Π be the set of on-going processes.

- `p-do-all`$(\mathcal{A})$, where $\mathcal{A}$ is a set of acts. `p-do-all` reduces to

```
cascade(do-all({Stop(p) |p ∈ Π ∩ A⊥}),
        do-all(A⊤ − Π}),
        p-do-all(A⊥)).
```

[5]Note that it just *initiates* them in arbitrary order. Once initiated, the acts may run in parallel.

That is, if any of the acts to be prioritized is already being performed, the agent first stops it if it has a low priority (the `Stop` act). It then performs acts with top priorities unless they are already on-going. Finally, the agent performs a `p-do-all` of the acts with low priorities (including those that were stopped). Thus, `p-do-all` provides a mechanism for performing acts according to their priorities while taking into account the set of on-going processes.

6 Prioritized Forward Inference

As pointed out in Section 4, perception is modeled by assertion with forward inference. When the body senses something (for example, being empty-handed or the battery going low), it communicates such information to the mind (GLAIR's knowledge level) in the appropriate form. Forward inference may activate believed `when-do`'s causing a reaction to be scheduled for performance or a pending cascade to move on. However, to handle interrupts, one needs more than such a reactive behavior. For example, consider an agent performing the sequence ⟨`PickUp-Block`, `Goto-Table`, `Put-Down-Block`⟩. The agent has just picked-up the block and it senses that it is holding it. At the same time, it also sense that the battery is low. In this case, two assertions will be made with forward inference, one asserting that the agent is holding the block and the other that the battery is low. This will accordingly cause two acts to be scheduled for execution: going to the recharging station and the pending cascade. To react appropriately to such a situation the agent needs to reason about the priorities of these acts. To model this kind of conscious reaction, we introduced a new mode of forward inference, one that may be called *prioritized forward inference* (PFI). Normal forward inference results in conclusions made by the agent and may also result in some acts being scheduled on an act stack, Σ. These acts are then initiated according to their arbitrary order in Σ. With PFI, on the other hand, all the scheduled acts are replaced by a single `p-do-all`. More precisely, where Π is the set of on-going processes,

$$\Sigma \longleftarrow \{\texttt{p-do-all}(\Sigma \cup \Pi)\}$$

Thus, not only will the agent reason about the priorities of the scheduled acts, but will also take all the on-going processes into account, in case it needs to suspend any of them.

7 Conclusions

A cognitive agent should be capable of reasoning about the priorities of its actions in order to appropriately recover from errors and handle interrupts. The system we presented has a number of advantages over other proposed models in the symbolic AI literature.

1. It provides a general mechanism for prioritized acting using the `p-do-all` construct. Interrupt handling comes out smoothly as a special case of prioritized acting (when Π is non-empty).
2. The `cascade` control act models the agent's expectation of failure and provides the bookkeeping required for resumption in case a cascade is interrupted.
3. Knowledge about priorities of acts may be given to the agent in natural language during an interaction with an operator. This may happen on-line, while the agent is acting.
4. Priorities are context-sensitive, changing according to various conditions in the environment.
5. Given the definition of the $>_p$ relation and `p-do-all`, the agent may interleave the execution of two cascades according to the priorities of acts in each.

8 Acknowledgements

The authors thank the members of the SNePS Research Group of the University at Buffalo for their support and comments on the work reported in this paper. This work was supported in part by ONR under contract N00014-98-C-0062.

References

[1] Henry Hexmoor, Johan Lammens, and Stuart C. Shapiro. Embodiment in GLAIR: a grounded layered architecture with integrated reasoning for autonomous agents. In Douglas D. Dankel, II and John Stewman, editors, *Proceedings of The Sixth Florida AI Research Symposium (FLAIRS 93)*, pages 325–329. The Florida AI Research Society, April 1993.

[2] Henry Hexmoor and Stuart C. Shapiro. Integrating skill and knowledge in expert agents. In P. J. Feltovich, K. M. Ford, and R. R. Hoffman, editors, *Expertise in Context*, pages 383–404. AAAI Press/MIT Press, Menlo Park, CA / Cambridge, MA, 1997.

[3] Stuart C. Shapiro and William J. Rapaport. SNePS considered as a fully intensional propositional semantic network. In N. Cercone and G. McCalla, editors, *The Knowledge Frontier*, pages 263–315. Springer-Verlag, New York, 1987.

[4] Stuart C. Shapiro and William J. Rapaport. The SNePS family. *Computers and mathematics with applications*, 23(2–5):243–275, 1992. Reprinted in F. Lehman, ed. *Semantic Networks in Artificial Intelligence*, pages 243–275. Pergamon Press, Oxford, 1992.

[5] Stuart C. Shapiro and the SNePS Implementation Group. *SNePS 2.5 User's Manual.* Department of Computer Science and Engineering, State University of New York at Buffalo, 1999.

[6] Henry Hexmoor. *Representing and Learning Routine Activities.* PhD thesis, Department of Computer Science, State Uni-

versity of New York at Buffalo, Buffalo, NY, 1995.

[7] Stuart C. Shapiro. Embodied Cassie. In *Cognitive Robotics: Papers from the 1998 AAAI Fall Symposium*, pages 136–143, Menlo Park, CA, 1998. AAAI Press. Technical report FS-98-02.

[8] Stuart C. Shapiro. The CASSIE projects: An approach to natural language competence. In *Proceedings of the 4th Portugese Conference on Artificial Intelligence*, pages 362–380, Lisbon, Portugal, 1989. Springer-Verlag.

[9] Stuart C. Shapiro and William J. Rapaport. An introduction to a computational reader of narratives. In *Deixis in Narrative: A Cognitive Science Perspective*, pages 79–105. Lawrence Erlbaum Associates, Inc., Hillsdale, NJ, 1995.

[10] Łucja M. Iwańska and Stuart C. Shapiro, editors. *Natural Language Processing and Knowledge Representation: Language for Knowledge and Knowledge for Language.* AAAI Press/The MIT Press, Menlo Park, CA, 2000.

[11] Michael Georgeff and Amy Lansky. Reactive reasoning and planning. In *Proceedings of the 6th National Conference on Artificial Intelligence*, pages 677–682, Los Altos, CA, 1987. Morgan Kaufmann.

[12] Murray Shanahan. Reinventing shakey. In *Cognitive Robotics: Papers from the 1998 AAAI Fall Symposium*, pages 125–135, Menlo Park, CA, 1998. AAAI Press. Technical report FS-98-02.

[13] Ernest Davis. Semantics for tasks that can be interrupted or abandoned. In James Hendler, editor, *The 1st International Conference on Artificial Intelligence Planning Systems (AIPS '92)*, pages 37–44, San Francisco, CA, 1992. Morgan Kaufmann.

[14] Hector Levesque, Ray Reiter, Yves Lespérance, Fangzhen Lin, and Richard Scherl. GOLOG: A logic programming language for dynamic domains. *The Journal of Logic Programming*, 31(1–3):59–83, 1997.

[15] Giuseppe De Giacomo, Yves Lespérance, and Hector Levesque. Reasoning about concurrent execution, prioritized interrupts, and exogenous actions in the situation calculus. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 1221–1226, San Francisco, CA, 1997. Morgan Kaufmann.

[16] Haythem O. Ismail and Stuart C. Shapiro. Cascaded acts: Conscious sequential acting for embodied agents. Technical Report 99-10, Department of Computer Science and Engineering, State University of New York at Buffalo., 1999.

[17] Deepak Kumar and Stuart C. Shapiro. Acting in service of inference (and vice versa). In Douglas D. Dankel, II, editor, *Proceedings of the Seventh Florida Artificial Intelligence Research Symposium*, pages 207–211, St. Petersburg, FL, May 1994. The Florida AI Research Society.

[18] Deepak Kumar. The SNePS BDI architecture. *Decision Support Systems*, 16:3–19, 1996.

[19] Deepak Kumar and Stuart C. Shapiro. The OK BDI architecture. *International Journal on Artificial Intelligence Tools*, 3(3):349–366, March 1994.

[20] Haythem O. Ismail and Stuart C. Shapiro. Two problems with reasoning and acting in time. In A. G. Cohn, F.Giunchiglia, and B. Selman, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Seventh International Conference (KR 2000)*, pages 355–365, San Francisco, CA, 2000. Morgan Kaufmann.