A SNePS Agent for Unexploded Ordnance Disposal
Progress Report

Stuart C. Shapiro and Haythem O. Ismail
Department of Computer Science and Engineering
State University of New York at Buffalo
226 Bell Hall
Buffalo, NY 14260-2000
Phone: (716) 645-3180
E-mail: {shapiro | hismail}@cse.buffalo.edu
August 27, 1999

---

1 Introduction

In this document, we present recent developments in the design of the knowledge level of the UXO disposal agent. Those developments enhance the interrupt handling model that was presented in our 5/31/1999 report. Although there are still problems to be solved, we have managed to solve two of the problems pointed out in the last report in addition to discovering and solving other problems. Section 2 provides a brief review of the main ideas underlying the interrupt handling system. Section 3 describes changes to the SNePS act scheduler. Section 4 discusses enhancements of the priority system and the prioritized acting procedure. Section 5 introduces a problem that we discovered with mental acts and outlines our current solution. Finally, Section 6 discusses problems and future work. Appendix A contains a demonstration of the system.

2 Interrupt Handling

In this section, we briefly review the main contents of our last (5/31/1999) report. In that report we explained in detail how the SNePS system was used to allow the UXO disposal agent to handle interrupts. By outlining the main ideas presented therein, we lay the ground to a detailed discussion of recent enhancements of the system.

1. A theory of interrupt handling subsumes one for error recovery. An error is just a special kind of an interrupt, one resulting from the unexpected situation of the failure of one of the agent's acts.

2. We distinguish two types of interrupts. First, an interrupt may occur during performing a sequence of acts. In particular, the agent has finished performing one act in the sequence and is about to perform the next when the interrupt occurs. Second, the interrupt may occur while the agent is in the midst of performing a primitive act. In the first case, the agent needs to merely revise its intentions regarding what to do next. In the second case, the agent may need to stop what it is doing to handle the interrupt.

3. Priorities define a partial ordering over acts. Formally, the following schema is used to represent knowledge about priorities.

   - *Holds*(*p-higher*(a_1, a_2), t)

   The above form means that at time t, a_1 has priority over a_2. The important point here is that priorities are dependent on the over-all situation.

4. A special mode of forward inference, prioritized forward inference (PFI), is used to model perception and bodily feedback. With PFI, actions that get scheduled as reactions to sensory input are performed in order of their priorities. This is achieved using the control act `p-do-all`.

   - `p-do-all`(*object1*), where *object1* is a set of act terms. `p-do-all` reduces to `cascade`(`do-all`(*MAX(object1)*), `p-do-all`(*object1*$-$*MAX(object1)*)), where *MAX(object1)* = {a $\in$ *object1* | there is no â $\in$ *object1* such that *Holds*(*p-higher*(â, a), **NOW**)}.[1]

   Note that this only takes care of the first type of interrupts.

5. Interrupting primitive acts is achieved by using the control act `interrupt`. Basically, `interrupt` results in stopping all the currently on-going acts (those pointed to by some of the modality pointers) and scheduling them together with the interrupting act.

   - `interrupt`(*todo*), where *todo* is an act term. `interrupt` reduces to

     `cascade`(`do-all`({`Stop`(*μ) $|\mu \in \mathcal{M}$}), `p-do-all`({*$\mu|\mu \in \mathcal{M}$} $\cup$ {*todo*}))

     where *todo* is the interrupting act, $\mathcal{M}$ is the set of modality pointers, and *μ denotes the term pointed to by μ.

Four problems were pointed out regarding the above interrupt handling mechanism (see Section 7 of our 5/31/1999 report). Of these, two were resolved given the new enhancements to the system. In what follows, we discuss how this was achieved.

[1] `do-all` is a control act that initiates a set of acts in some nondeterministic order. `cascade` is one that controls the execution of sequences of acts. See our previous report for more details.

3 The New Scheduler

Given the above discussion, there seems to be three unsatisfying issues regarding the proposed interrupt handling mechanism.

1. There are two different mechanisms for handling primitive and composite acts.
2. The agent always stops everything that it is doing when an interrupt occurs. Those on-going acts with high priorities should be allowed to continue to completion.[2]
3. It is not clear when exactly to use `interrupt`. The act is used as a reaction to some situation that requires the agent to stop everything that it is doing and take some appropriate action. But then all situations that require the agent to react should make use of `interrupt`. In general, whenever the agent is doing something and it decides that it is appropriate to perform some other act (whether because it was told to do so or because it needs to react to some situation) it should *reason* about whether to interrupt what it is currently doing. Since this is a general requirement, it would be better to build interrupt handling into the acting system rather than require explicit use of `interrupt` in the knowledge base.

Solving the above problems was achieved be revising both PFI and the procedural semantics of `p-do-all`. In this section, we discuss the revised PFI procedure and defer the discussion of `p-do-all` to the next section. As pointed out in Section 2, PFI results in all scheduled acts being performed in order of their priorities. More specifically, the acts scheduled on the act stack, Σ, are replaced by a `p-do-all` of those acts. That is:

$\Sigma \longleftarrow \{$`p-do-all`$(\Sigma)\}$

To build interrupt handling into the act scheduler, we revised PFI so that the contents of Σ are replaced by a `p-do-all` of all the scheduled acts in addition to those that are currently being performed.

$\Sigma \longleftarrow \{$`p-do-all`$(\Sigma \cup \Pi)\}$, where Π is the set of on-going acts (processes).

This way, whenever the agent is about to act (while in PFI mode), it has to reason about (i) what to do first and (ii) whether to interrupt its on-going activities.

[2] This is the fourth problem pointed out in our last report (page 12).

This solves the first and third problems above. We no longer need the control act `interrupt` since its effects have been built into the acting system. In addition, PFI now provides a single, albeit more complicated, mechanism for handling the two types of interrupts that we distinguish. Solving the second problem mentioned above is based on the new procedural semantics of `p-do-all`.

4 Prioritized Acting

According to Section 2, `p-do-all` reduces to a cascade of a `do-all` of those acts with highest priority followed by a `p-do-all` of the rest of the acts. Obviously, this presupposes that all of these acts are not in progress. Given the revised PFI procedure, we needed to revise `p-do-all` so that it results in the appropriate behavior in case any of its argument acts is in progress. Before getting into this, however, let us first discuss two important revisions of `p-do-all` and the priority system.

4.1 Priorities

Consider the following problem.[3] The agent is performing an iterative act of searching for, picking up, and dropping UXOs. This loop is cascaded so that the agent starts picking up when it has finished searching, starts dropping when it has finished picking up, and starts searching again when it has finished dropping. Suppose that the battery goes low just when the agent is about to drop a UXO (or actually is dropping one). This will result in a `p-do-all` with the acts of dropping the UXO and recharging the battery to be performed. Dropping the UXO should have higher priority and therefore should be done first. However, once completed, two pending cascades would be activated: one with `Recharge-battery` and another with the search, pick up, and drop cycle. These two acts are arbitrary and emerge from the particular situation. The problem is that arbitrary cascades are not prioritized with respect to each other and the agent may end up searching instead of recharging the battery.

The way out of this problem is to define a more complicated version of the priority relation.

Definition 4.1 *Let a_1 and a_2 be two distinct acts. $a_1 >_p a_2$ (read, a_1 has higher priority over a_2) iff:*

1. $a_1 = \mathit{cascade}(a_i, \ldots, a_{i+n})$ *and* $a_i >_p a_2$,

[3] The third problem in our 5/31/1999 report (page 12).

2. $a_2 = cascade(a_j, \ldots, a_{j+m})$ *and* $a_1 >_p a_j$, *or*

3. Holds(p-higher(a_1, a_2), *NOW) *is deducible.*

Accordingly, the relation $>_p$ holds between two cascades, c_1 and c_2, if it holds between their first elements. The base case of the recursion is the explicit assertion of priorities (in terms of *p-higher*) among acts. However, the above does not say that c_1 should be completed before starting to perform c_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 4.2 *Let A be a set of acts. Define the set $A_\top$ as follows. An act $a \in A_\top$ iff:*

1. *$a \in A$, a is a cascade, and for every $\grave{a} \in A$,* Holds(p-higher(a, $\grave{a}$), *NOW) *is deducible;*

2. *$a \in A$, a is not a cascade, and there is no $\grave{a} \in A$ such that $\grave{a} >_p a$; or*

3. *$c = cascade(a, \ldots, a_n) \in A$ and there is no $\grave{a} \in A$ such that $\grave{a} >_p c$.*

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

Definition 4.3 *Let A be a set of acts. Define* $A_\perp = (A - A_\top) \cup \{cascade(a_{2_i}, \ldots, a_{n_i}) \mid cascade(a_{1_i}, a_{2_i}, \ldots, a_{n_i}) \in A$ *and* $a_{1_i} \in A_\top\}$

Informally, a `p-do-all` with a set of acts A reduces to a cascade of `do-all`($A_\top$) followed by `p-do-all`($A_\perp$).[4] Going back to the problem discussed at the beginning of this section, and assuming that *Holds*(*p-higher*(`Recharge-battery`, `Search`), *NOW) is deducible, we have the following situation:

- A = {cascade(`Recharge-battery`), cascade(`Search`, `Pick-up`, `Drop`)}.
- $A_\top$ = {`Recharge-battery`}.
- $A_\perp$ = {cascade(`Search`, `Pick-up`, `Drop`)}.[5]

Thus the agent would exhibit the correct behavior; recharging the battery and then resuming what it was doing before.

[4] But see below for some subtle differences.

[5] Note that empty cascades are not allowed by Definition 4.3.

4.2 The Revised p-do-all

Consider a situation similar to the above one. The agent is performing the cascade cascade(Drop, Goto(*Safe-zone*), Talk-to(*Bill*)). It starts dropping the UXO that it is holding while having the belief that, when it is done, it will perform the rest of the cascade. While in the midst of dropping the UXO, the battery goes low. This results in the performance of p-do-all({Recharge-battery, Drop}) Assuming that dropping the UXO has higher priority than recharging the battery, the agent believes that when it is done it will perform p-do-all({Recharge-battery}). Now the agent finishes dropping the UXO thereby scheduling two acts with PFI, namely cascade(Goto(*Safe-zone*), Talk-to(*Bill*)) and p-do-all({Recharge-battery}). The second of these is an arbitrary p-do-all. and in the general case may contain more than one act. How can a p-do-all embedded within a p-do-all be compared with other acts? The intuitive answer is to first *expand* the embedded p-do-all and prioritize the acts in the resulting set.

More precisely, given a set of acts, A, as an argument to p-do-all, the set that actually gets prioritized is defined as follows.

Definition 4.4 *Let A be a set of acts. Define* $A^E = A - \{p\text{-}do\text{-}all(S_i) \mid p\text{-}do\text{-}all(S_i) \in A\} \cup \bigcup_i (S_i)_{\texttt{p-do-all}(S_i) \in A}$

We are now ready to give the precise semantics of the revised p-do-all.

- p-do-all(*object1*), where *object1* is a set of act terms. p-do-all reduces to

  cascade(do-all({Stop(p) $|p \in \Pi \cap object1^E_\perp$}),
  do-all($object1^E_\top - \Pi$),
  p-do-all($object1^E_\perp$)).

That is, first the agent stops all on-going acts with low priorities, then it performs acts with top priorities unless they are already on-going, and then performs a p-do-all of the acts with low priorities. Note that all of this is based on an expanded set of acts according to Definition 4.4. Evidently, the agent only stops those acts with low priorities whereas top priority on-going acts are allowed to continue uninterrupted. This solves the second problem presented in Section 3.

5 The Mental Acts Queue

Consider the situation described in Section 4.2. The agent is performing the cascade cascade(Drop, Goto(*Safe-zone*), Talk-to(*Bill*)). Let us refer to this cascade

as c_1. Assume that the agent starts performing c_1 while it is in the field (Z1). To drop the UXO it needs to achieve the precondition for dropping; being in the drop-off zone (Z4). This is done be performing the cascade `cascade(Goto(`$Z4$`), Drop)`.[6] Let us call this cascade c_2. Before starting to perform c_2, the agent senses that the battery is low. This results in scheduling a `p-do-all` of c_2 and `Recharge-battery`. Assuming that c_2 has higher priority over recharging the battery, the agent sets out to perform `cascade(`c_2`, Recharge-battery)`.[7] Let us refer to that as c_3. To perform c_3, the agent needs to start performing c_2 and to form the belief that when the goal of c_2 has been achieved it should perform `Recharge-battery`. What is the goal of c_2? According to Definition 3.1 in our last report (page 4), it is achieving the state `complete(`c_2`)`. This state gets asserted by appending the mental act `believe(complete(`c_2`))` to the end of c_2. That is, c_2 expands to `cascade(Goto(`$Z4$`), Drop, believe(complete(`c_2`)))`.

Now the agent reaches Z4. At this point, it would be useful to inspect the agent's intentions. In particular, it has the following beliefs.

- `when-do(`*Empty-handed*`, cascade(Goto(`*Safe-zone*`), Talk-to(`*Bill*`)))`.
- `when-do(`*Empty-handed*`, believe(complete(`c_2`)))`.

The first form represents the agent's intention to continue c_1 after dropping the UXO. The second represents its intention to continue c_2. Once the agent finishes dropping the UXO and becomes empty-handed, the two acts `cascade(Goto(`*Safe-zone*`), Talk-to(`*Bill*`))` and `believe(complete(`c_2`))` get scheduled. These two acts are then performed in order of their priorities. However, this seems very unsatisfying. The only reason the mental act was introduced is to enable scheduling `Recharge-battery` once the agent has finished dropping the UXO. It is the priority of `Recharge-battery` that should be compared to that of the rest of c_1, not the priority of the mental act.

One way out of this problem is to introduce a rule to the affect that mental acts have higher priority over all other acts. This might indeed solve the problem. However, it seems awkward for the agent to *reason* whether it should *believe* before, say, search. If the agent has the intention to believe some proposition, it should do so once the appropriate conditions are met (in the above example, having dropped the UXO). This requires us to build the higher priority of mental acts into the acting system rather than explicitly represent it in the knowledge base. This was done by introducing a special queue, Q, for mental acts. The acting system only starts processing the contents of Σ, the non-mental acts stack, when it has processed all the contents of Q. This way, mental acts will be performed before

[6] See Section 5 of our 5/31/1999 report.

[7] A `do-all` or a `p-do-all` with one act reduces to that act.

other physical and control acts and will not be considered in priority checks that are only applicable to the contents of Σ.

To go back to our example, the act `believe(complete(`c_2`))` shall be performed resulting in `Recharge-battery` to be scheduled with the rest of c_1 on Σ. A `p-do-all` of these two will be performed thereby comparing the correct priorities.

6 Problems and Future Work

1. In the current state of the system, the agent *always* resumes what it was doing after handling an interrupt. However, it is not always appropriate for the agent to resume what it was doing. For example, if one of its motors needs service, the agent should completely abort what it is currently doing.

2. Currently the agent cannot correctly resume what it was doing in some cases. In particular, consider the agent being near a UXO and about to pick it up when the battery goes low. Assuming that recharging the battery has higher priority than picking up the UXO, the agent would go to the recharge station and recharge the battery. Given the way the system currently works (i.e., the use of PFI), the agent would next attempt to pick up the UXO. However, it is no longer near the UXO and therefore cannot pick it up. One might think that the problem can be solved by having the state of being near a UXO a precondition for picking it up and that this would cause the agent to first achieve being near the UXO. However, the problem is that what the agent is trying to do is to pick up a particular UXO, the one that it was near when the battery went low. Since UXOs are indistinguishable and the agent does not have a record of where it was before, it cannot go back near that particular UXO. The solution is to start searching for UXOs again, instead of trying to pick up some specific one. That is, the agent should resume what it was doing but not exactly at the point where it left off.

3. We have started using multi-processing in the implementation of the agent. Currently, multi-processing features are only used in simulating some aspects of the agent's environment. In particular, we use a different process to simulate the explosion of a UXO. The simulated agent sets a charge on the UXO and thereby initiates a process that simulates the explosion. In the future, we intend to make more use of multi-processing since it provides a more accurate model of the environment and implementation of the actual robot.

4. We need to test the system with different types of interrupts. The only type of interrupt that we have implemented so far is the battery going low.

A Ascii World Demonstration

The following is a demonstration of the high level behaviors `Gather-Objects` and `Blow-up Objects` in the ASCII world. Sentences surrounded by asterisks are the output of the world simulation. Sentences preceded by semi-colons are explanatory comments. Otherwise, it is the agent explaining what it is doing. In both cases, the first sentence, following the prompt (:), is the natural language input to the system. For this demonstration, we manually set the field to contain only four objects.

```
;;;------The High-Level Behavior: Gather-Objects------


 : Clear the field.
 I am going to Z1.
**The robot is going to WORLD:Z1.**
**The robot is in WORLD:Z1 at the point: (150.00, 5.00).**
 I went to Z1.
 I am in Z1.
 I am searching.
**The robot is searching for a UXO . . .**
**Object found at: (204.06, 22.04).**
**The robot is going near the object (WORLD::ORANGE 204.0638 22.035358) . . .**
**The robot is looking at the object (WORLD::ORANGE 204.0638 22.035358).**
**The robot is near the object (WORLD::ORANGE 204.0638 22.035358).**
**The robot is going to examine object.**
**OBJECT FOUND IS A UXO.**
 I searched.
 I am near a UXO.
 I am picking up the UXO.
**The robot is picking up the UXO.**
 I picked up the UXO.
 I am holding the UXO.
 I am turning to Z4.
**The robot is looking towards WORLD:Z4.**
 I turned to Z4.
 I am looking at Z4.
 I am going to Z4.
**The robot is going to WORLD:Z4.**
**The robot is in WORLD:Z4 at the point: (35.81, 3.69).**
 I went to Z4.
```

```
 I am in Z4.
 I am dropping the UXO.
**The robot is dropping the UXO.**
 I dropped the UXO.
 I am empty handed.
 I am turning to Z1.
**The robot is looking towards WORLD:Z1.**
 I turned to Z1.
 I am looking at Z1.
 I am going to Z1.
**The robot is going to WORLD:Z1.**
**The robot is in WORLD:Z1 at the point: (38.00, 3.69).**
 I went to Z1.
 I am in Z1.
 I am searching.
**The robot is searching for a UXO . . .**
**Object found at: (49.88, 109.08).**
**The robot is going near the object (WORLD::WHITE 49.881214 109.07636) . . .**
**The robot is looking at the object (WORLD::WHITE 49.881214 109.07636).**
**The robot is near the object (WORLD::WHITE 49.881214 109.07636).**
**The robot is going to examine object.**
**OBJECT FOUND IS NOT A UXO.**
**The robot is searching for a UXO . . .**
**Object found at: (84.77, 125.84).**
**The robot is going near the object (WORLD::ORANGE 84.7668 125.843216) . . .**
**The robot is looking at the object (WORLD::ORANGE 84.7668 125.843216).**
**The robot is near the object (WORLD::ORANGE 84.7668 125.843216).**
**The robot is going to examine object.**
**OBJECT FOUND IS A UXO.**
 I searched.
 I am near the UXO.
 I am picking up the UXO.
**The robot is picking up the UXO.**
 I picked up the UXO.
 I am holding the UXO.
 I am turning to Z4.
**The robot is looking towards WORLD:Z4.**
 I turned to Z4.
 I am looking at Z4.
 I am going to Z4.
**The robot is going to WORLD:Z4.**
**The robot is in WORLD:Z4 at the point: (19.69, 30.14).**
```

```
**<<<THE BATTERY IS LOW>>>.**
```

;;;The battery goes low while the agent is about to drop the UXO.
;;;This is similar to the situation discussed in Section 4.1.
;;;Also similar to that discussed in Section 5.

```
 I went to Z4.
 I am in Z4
and a battery is low.
 I am dropping the UXO.
**The robot is dropping the UXO.**
 I dropped the UXO.
 I am empty handed.
```

;;;Dropping the UXO has higher priority over recharging the battery if in Z4.

```
 I am turning to Z3.
**The robot is looking towards WORLD:Z3.**
 I turned to Z3.
 I am looking at Z3.
 I am going to Z3.
**The robot is going to WORLD:Z3.**
**The robot is in WORLD:Z3 at the point: (19.69, -5.00).**
 I went to Z3.
 I am in Z3.
 I am recharging the battery.
**The robot is recharging the battery.**
 I recharged the battery.
 the battery is full.
 I am talking to you.
```

;;;Correctly prioritized the two cascades of recharging the battery and of the main
;;; gathering loop (see Sections 4.1 and 5).
;;;The on-going talking was stopped while recharging and now resumes.

```
 I am turning to Z1.
**The robot is looking towards WORLD:Z1.**
 I turned to Z1.
 I am looking at Z1.
 I am going to Z1.
```

```
**The robot is going to WORLD:Z1.**
**The robot is in WORLD:Z1 at the point: (38.00, 5.00).**
 I went to Z1.
 I am in Z1.
 I am searching.
**The robot is searching for a UXO . . .**
**Object found at: (30.00, 151.88).**
**The robot is going near the object (WORLD::WHITE 30.004522 151.88202) . . .**
**The robot is looking at the object (WORLD::WHITE 30.004522 151.88202).**
**The robot is near the object (WORLD::WHITE 30.004522 151.88202).**
**The robot is going to examine object.**
**OBJECT FOUND IS NOT A UXO.**
**The robot is searching for a UXO . . .**
 Z1 is cleared.
 I am turning to Z2.
**The robot is looking towards WORLD:Z2.**
 I turned to Z2.
 I am looking at Z2.
 I am going to Z2.
**The robot is going to WORLD:Z2.**
**The robot is in WORLD:Z2 at the point: (38.00, -5.00).**
 I went to Z2.
 I am in Z2.

;;;------The High-Level Behavior: Blow-up Objects------

 : Blow up the UXOs.
 I am going to Z1.
**The robot is going to WORLD:Z1.**
**The robot is in WORLD:Z1 at the point: (150.00, 5.00).**
 I went to Z1.
 I am in Z1.
 I am searching.
**The robot is searching for a UXO . . .**
**Object found at: (21.10, 317.18).**
**The robot is going near the object (WORLD::WHITE 21.103125 317.18237) . . .**
**The robot is looking at the object (WORLD::WHITE 21.103125 317.18237).**
**The robot is near the object (WORLD::WHITE 21.103125 317.18237).**
**The robot is going to examine object.**
**OBJECT FOUND IS NOT A UXO.**
**The robot is searching for a UXO . . .**
**Object found at: (187.23, 327.45).**
```

```
 **The robot is going near the object (WORLD::WHITE 187.23366 327.44772) . . .**
 **The robot is looking at the object (WORLD::WHITE 187.23366 327.44772).**
 **The robot is near the object (WORLD::WHITE 187.23366 327.44772).**
 **The robot is going to examine object.**
 **OBJECT FOUND IS NOT A UXO.**
 **The robot is searching for a UXO . . .**
 **Object found at: (36.72, 347.09).**
 **The robot is going near the object (WORLD::ORANGE 36.72171 347.08948) . . .**
 **The robot is looking at the object (WORLD::ORANGE 36.72171 347.08948).**
 **The robot is near the object (WORLD::ORANGE 36.72171 347.08948).**
 **The robot is going to examine object.**
 **OBJECT FOUND IS A UXO.**
  I searched.
  I am near a UXO.
 **The robot has a charge.**
  I am placing a charge.
 **The robot is placing a charge on the UXO (WORLD::ORANGE
                                             36.72171
                                             347.08948).**
 **The charge is on the UXO.**
  I am empty handed.
  I placed the charge.
  the charge is on the UXO.
  I am hiding.
 **The robot is looking for a safe place. . .**
 **The robot is going to the point: (141.67, 346.41). . .**
 **The robot is at a safe place.**
 **<<<THE BATTERY IS LOW>>>.**
```

;;;While waiting for the explosion, the battery goes low.
;;;Note that continuing to wait should have higher priority.

```
  I hid.
  I am safe
and the battery is low.
  I am turning to Z3.
 **The robot is looking towards WORLD:Z3.**
  I turned to Z3.
  I am looking at Z3.
 **!!BANG!!**
  I am going to Z3.
 **The robot is going to WORLD:Z3.**
```

```
**!!BANG!!**
**!!BANG!!**
```

;;;Note how the two processes (the explosion and the agent's going to Z3) are
;;;interleaved in the simulation.

```
**!!BANG!!**
**!!BANG!!**
**!!BANG!!**
**!!BANG!!**
**!!BANG!!**
**The robot is in WORLD:Z3 at the point: (32.00, -5.00).**
 I went to Z3.
 I am in Z3.
 I am recharging the battery.
**The robot is recharging the battery.**
 I am sensing an explosion.
```

;;;Even though it has sensed the explosion, it does not interrupt recharging the
;;;battery (see Section 4.2).

```
 I recharged the battery.
 the battery is full.
 I am talking to you.
```

;;;Now it reacts to sensing the explosion.

```
 I am turning to Z1.
**The robot is looking towards WORLD:Z1.**
 I turned to Z1.
 I am looking at Z1.
 I am going to Z1.
**The robot is going to WORLD:Z1.**
**The robot is in WORLD:Z1 at the point: (38.00, 5.00).**
 I went to Z1.
 I am in Z1.
 I am searching.
**The robot is searching for a UXO . . .**
**Object found at: (172.34, 368.48).**
**The robot is going near the object (WORLD::WHITE 172.33784 368.4796) . . .**
```

```
**The robot is looking at the object (WORLD::WHITE 172.33784 368.4796).**
**The robot is near the object (WORLD::WHITE 172.33784 368.4796).**
**The robot is going to examine object.**
**OBJECT FOUND IS NOT A UXO.**
**The robot is searching for a UXO . . .**
 Z1 is cleared.
 I am turning to Z2.
**The robot is looking towards WORLD:Z2.**
**<<<THE BATTERY IS LOW>>>.**
 I turned to Z2.
 I am looking at Z2.
 I am going to Z2.
**The robot is going to WORLD:Z2.**
**The robot is in WORLD:Z2 at the point: (170.60, -5.00).**
 I went to Z2.
 I am in Z2
and the battery is low.
 I am turning to Z3.
**The robot is looking towards WORLD:Z3.**
 I turned to Z3.
 I am looking at Z3.
 I am going to Z3.
**The robot is going to WORLD:Z3.**
**The robot is in WORLD:Z3 at the point: (32.00, -5.00).**
 I went to Z3.
 I am in Z3.
 I am recharging the battery.
**The robot is recharging the battery.**
 I recharged the battery.
 the battery is full.
 I am talking to you.
```