Towards Believable Intelligent Virtual Agents with StateFull Hierarchical Reactive Planning

T. Plch

Charles University Prague, Faculty of Mathematics and Physics, Prague, Czech Republic.

Abstract. Intelligent Virtual Agents (IVAs) deployed within virtual worlds are required to exhibit believable rational behavior. The reactive action selection mechanism represents a popular choice for most Artificial Intelligence (AI) architectures, mainly due to simple implementation (e.g. Finite State Machines, If-Then rules etc.) and timely execution. We identified several problems of the commonly utilized Hierarchical Reactive Planning (HRP) approach summarized into three classes: a) intentions, b) planning, and c) transitional behaviors. We extended the HRP approach by a StateFull semantic layer on top of the reactive If-Then rule set thus creating StateFull HRP (SF HRP) architecture. Our architecture's goals are twofold 1) aid action selection to maintain believable behavior (e.g. transitions, plan aftereffect etc.) and 2) add semantic data to reactive plans on which reasoning can be performed. Our paper presents the SF HRP architecture and the possible use of additional data for object driven reasoning. We implemented a simple proof of concept prototype on which we base our conclusions.

Introduction

The term of intelligent virtual agent (IVA) denotes a software agent situated within a dynamic, complex, real–time, and unpredictable 3D virtual environment. The requirements on IVA's rationality [*Russel et al,* 2003] and believable behavior [*Loyall*, 1997] increases with ever growing realism of virtual environments they inhabit. The *Action Selection Mechanism* (ASM), which produces the IVA's behavior, is one of the dominating topics for agent Artificial Intelligence (AI) research.

The main goal of the ASM is to choose actions by which the agent can influence its environment and satisfy its goals, which are either predefined by the AI designer or a product of the agent's reasoning. The ASMs for IVAs can be divided into two basic groups – a) *deliberative* and b) *reactive*, based on the approach they choose the next action. Classical planning [*Ghallab et al,* 2004] is the typical representative of the deliberative paradigm. The main idea is to produce a sequence of actions creating a transition from the current to the goal state, based on the action's preconditions and effects.

On the other hand, the reactive approach, where Finite State Machines (FSM) [*Champandard,* 2003] and Hierarchical Reactive Planning (HRP) [*Bryson,* 2001] are key representatives, is build around the idea of selecting an action based on the current context the agent periodically perceives (i.e. at every AI engine update). The main benefits of this approach are the IVA's responsiveness within dynamic environments and the implementation's simplicity. However, the resulting behavioral patterns are repetitive and schematic.

Our paper's focus is on the topic of reactive planning, especially the Hierarchical Reactive Planning (HRP) variant. Based on the computation power available for AI engines within most simulation, it represents a very capable candidate for believably behaving IVAs. However, Brom [2005] presented several problems of the HRP approach which degrade the believability of the resulting behavior. The goal of this paper is to present our StateFull HRP architecture which is designed to overcome the observed issues and further add modularity to the HRP concept. As a proof of concept, we implemented a simple simulation of our concept [*Plch*, 2009] and ran several simple scenarios illustrating the observed issues.

The paper's structure is as follows. The following section describes the concept of HRP in more detail. The third section is concerned with problems we used as guidelines for our design. The fourth section describes our StateFull HRP architecture in detail. The fifth section is concerned with our prototype implementation. The sixth section focuses on future work. The last section is concerned with the conclusion.

Hierarchical Reactive Planning (HRP)

The Hierarchical Reactive Planning (HRP) is one of the basic representatives, besides Finite State Machines (FSM), for the reactive action selection paradigm [*Bryson*, 2001]. The HRP's key component is the Reactive Plan (RP), being a set of *If-Then rules* (Table 1) ordered by priority. The RPs recursively form a hierarchical tree-like structure – a behavioral tree called *BE-Tree*, where nodes are RP and plan links (**Goto**) are edges. Leafs of the BE-Tree are execution primitives (e.g. atomic actions, action sequences etc.) which impact the state of the world (e.g. "grab shovel", "throw axe").

Priority	Precondition	Action
1	LowHealth	**Goto(**Medikit)
2	LowAmmo	**Goto**(Reload)
3	seeEnemy	Shoot
4	seeFriend	**Goto**(Greet)

→

Priority	Precondition	Action
1	See(Enemy)	Run
2	See(Medkit)	Grab
3	Not(See(Medkit))	Search

Table 1. Simple example of a HRP's Reactive Plan "Fight" and a subplan "Medkit"

The *Priority* represents an ordering within the RP on how important a rule is. The *Precondition* represents a logical trigger, which when satisfied, makes a rule a candidate for execution. The *Action* may be twofold – either an execution primitive (e.g. single atomic action, action sequence etc.), or a *plan link* – presented as **Goto** statement in Table 1 which moves the search for an execution primitive deeper into the BE-tree's hierarchical structure to a lower level RP (i.e. a subplan).

Formal representation

Similar to [*Bryson*, 2001], the HRP can be seen as a system choosing an action changing the current world state, without utilizing any look-ahead. The system can be formally specified as a triplet $\Sigma = (S, A, \gamma)$, where S is a recursively enumerable set of agent states (agent's perceptions of the world), A is a recursively enumerable set of actions, and γ is a function $S \times A \rightarrow \mathcal{P}(S)$, where $\mathcal{P}(S)$ is the power set of S. A world's state is similar to the classical planning's representation [*Ghallab et al*, 2004], as a set of positive literals (grounded, function-less predicates) in first order logic.

The hierarchy only helps the AI design process by decomposing the activities into a hierarchical structure. Formally, the hierarchy can be translated into a single RP, where every execution primitive is at the same level and its releaser is a conjunction of all releasers along the BE-tree's path from the root to the execution primitive in a leaf node. The priority is a concatenation of all the priorities over the path (i.e. priorities $1 \rightarrow 2 \rightarrow 5$ translates into a single priority 125). Every priority is padded with 0 (at the end of the string, e.g. 12 translates into 120), so all the priorities have the same length (i.e. the length of the deepest execution primitive). Therefore, hierarchical view is irrelevant, both representations (one level RP and HRP) are equivalent.

The HRP action can be formalized as a triplet *(Priority, Precondition, Act)*, where *Priority* $\in \mathbb{N}$, *Precontion* is a boolean expression in first order logic describing agent states and *Act* is an execution primitive that results in executing an action from A.

Action Selection Mechanism

The ASM for HRP is show in Algorithm 1. The main idea of the algorithm is to recursively search the BE-Tree structure until an atomic action can be executed. The rules are chosen simply by considering two factors – the priority of the rule and a holding trigger. The algorithm follows the plan links and iterates at every level of the hierarchy. It is noteworthy that problems resulting from the simplicity of ASM were covered in [*Plch et al*, 2010]. This ASM algorithm is executed at every sensory cycle to determine which action is to be executed.

```
ASM( root ):
(1) plan ← root
(2) preactive rules ← get all rules from plan with holding trigger, order by priority
(2) while (preactive rules not empty )
(2.1) active rule ← get and remove first of preactive rules
(2.2) case ( active rule )
(2.2.1) (is executable ) ret ← (execute (active rule) )
(2.2.2) (is plan link ) ret ← (ASM (active rule ))
(2.3) if ( ret is fail ) continue else return (ret)
(3) return fail
```

Algorithm 1. Action selection algorithm for HRP

Problems o f HRP

The main advantages of HRP are simplicity, ease of implementation and timely fashion of execution. These attributes made this approach a very popular one, used in computer games [*Isla*, 2005], military simulations [*Reidl et al*, 2006] and many others. The work presented in [*Brom*, 2005] and [*Plch*, 2009] studied the HRP techniques in further detail and discovered several key problems of the behavior resulting from HRP. These problems can be summarized into three classes: a) intentions, b) planning and deliberation, and c) transitional behavior.

Intentions

Based on the Belief-Desire-Intention [*Wooldridge,* 2002] model, an intention is a deliberative state, something the agent wants to achieve. An intention can be achieved by satisfying a set of goals, which are satisfied by reactive plans. This can be illustrated as follows: *intention →goals → plans*
Reactive planning provides only static structures with no capability to *add*, *remove* or *manage* intentions (with associated goals and plans), rendering the agent less receptive to newly introduced situations, able to solve only those who are hard-wired into his brain by the designer.

Planning and deliberation

In general, *planning* (in a classical sense) is what reactive planning tends to elude, keeping the agent responsive and reactive to its surroundings. However, a degree of planning is necessary to make the behavior of IVAs appear more intelligent and believable. Humans expect to see some deliberation and planning, to perceive intelligent behavior and see the IVA as more self-aware. We can divide the problems for reactive planning into categories: a) *pre-execution* b) *post-execution* c) *ordering*.

Pre-execution planning is tightly coupled with the ability to prepare the footing for successful plan execution (e.g. by timely acquisition of proper objects) or for the execution of a subplan (*deep preparation*) or a concurrent plan (*cross plan preparation*). Post-execution is concerned with smart cleaning up of objects. Effective ordering (*chaining*) of plan can even when breaking priority of execution, can lead to more effective execution patterns.

Transitional behavior

In dynamic environments, an executed reactive plan A could be interrupted and suspended by another plan B with higher priority (Figure 1). To be able to start plan B and later resume the plan A, it (plan A) has to be discontinued in a consistent way (e.g. hands of the agent are empty). In most cases, the observer awaits a smooth transition from one to the other behavior.

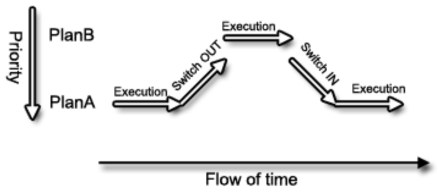

Figure 1. Simple example of PlanB with higher priority which suspends the PlanA with lower priority. Plan A is later resumed.

This topic was also addressed in [*Mikula*, 2006] and [*Bída et al*, 2011]. Expressing transitional behaviors can be almost impossible to achieve using plain HRP, mostly due to the fact, that plans do not have any knowledge of other plans that could succeed or surpass them.

A part of transitional behaviors is the *cleaning behavior* humans exhibit – when an activity is finished, items are put back to their respective places. The notion of cleaning up is tightly related to the knowledge of used objects by the current activity and other future activities that might make use of the objects in question. Creating a believable behavior using only HRP facilities could result in large complex plans and might not work at all.

StateFull Hierarchical Reactive Planning (SF HRP)

In this section, we present the main idea of the StateFull Hierarchical Reactive Planning (SF HRP) architecture. We extend the HRP's execution into a more structured form and base our design on the observation how actual Reactive Plans are designed by decomposing the task they address into phases by utilizing the priority of specific rule sets or creating subplans – *preparation, execution, cleanup*. We also adopt and extend the notion of *swiching* [*Mikula*, 2006], by introducing a specific set of phases addressing this issue. Thirdly, we add semantic data (*object list* and *object classes*) to RP on object needs and usage which can be utilized to perform reasoning and planning.

We introduce the StateFull Plan (SFP) which combines a Finite State Machine (FSM) integrating the RP's execution as a part of its workflow (Figure 2), thus maintaining backward compatibility with the HRP approach. We recognize the following phases: 1) initialization – used to prepare for plan execution (e.g. collects necessary objects), 2) execution – used to perform actions of carry out an activity (e.g. cut down trees), 3) termination – used after the plan terminates to perform necessary activities (e.g. empty agent's hands), 4) finish – used to evaluate the plan (e.g. choose objects to keep and for cleanup), 5) cleanup – used to perform cleanup (e.g. take the axe back to a shed), 6) switch out – used for consistently suspend the plan (e.g. drop objects in hands), 7) switch in – used to resume the suspended plan (e.g. find the axe and grab it), 8) switched – representing the suspended state (e.g. agent whistling while doing nothing), and 9) emergency – used for special occasions (e.g. someone throws an axe at the agent, he dodges). It is noteworthy, that minor changes to the rule formalism presented in [*Plch et al, 2010*] further augment our architecture. The transitions between these phases can be seen in Figure 3. The actual execution of the FSM is triggered when the SFP is chosen based on its priority and holding releaser in the BE-Tree's layer one level above.

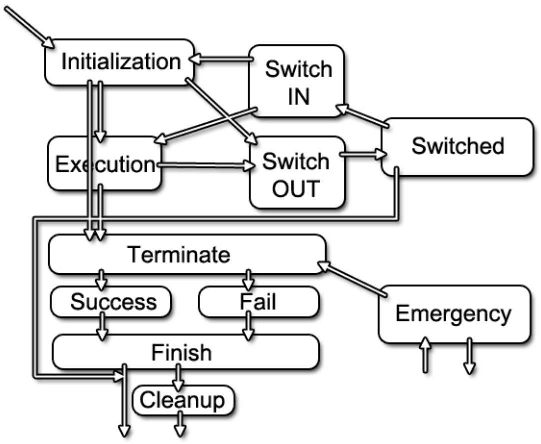

Figure 2. StateFull Plan's phases of execution. Boxes are phases and arrows are possible transitions

In most cases the SFP's execution is as follows – the SFP first enters the initialization phase, prepares for execution (e.g. collects objects) and afterwards enters the execution phase. During both of these phases, the workflow might be suspended by a higher priority SFP and the plan is switched out either in generic way (i.e. default switch out) or in respect to the suspending SFP (i.e. switch out based on a suspender ↔ suspended (plan) pair). After the suspended SFP (i.e. the switch in phase) is returned into execution, it continues and possibly terminates – either as success or fail. After performing some evaluation, the workflow finishes and possibly enters a Cleanup phase. This phase can be omitted or only partially executed, where the plan higher in the hierarchy takes responsibility for the cleanup. All mentioned phases can be realized either by a build-in code or script (e.g. C++ routines, LUA script) or by a simple SFP or RP.

These phases are distinct execution mechanisms and the basic idea of an implementation is to separate the above mentioned design phases into distinct entities (e.g. C++ classes) which have well defined transitions – i.e. the plan will not enter the execution phase, until all the objects necessary for successful execution are present. The plan also terminates in a consistent way (e.g. not needed objects are not in the agent's hands). If a transition behavior is required, the FSM above the RP allows for transparent (from the execution's point of view) suspending and resuming of the suspended activity, similarly to process switching in operating systems. The existence of a termination phase allows for additional execution reflecting the plan's success or fail (e.g. on success, the agent is happier, on fail it gets angry). Introducing the cleanup phase into the design, allows us to target the cleanup behavior not only of the currently executing plan, but also of plans below and above it in the hierarchy. The SFP at a given level is responsible for the cleanup of its own objects and of objects of plans below in the hierarchy. The SFPs above a given level can remove object from cleanup sets, based on requirements by other plans in other branches.

The notion of phases represented as entities (C++ classes) allows for extensive modularity - e.g. the IVA could change its plan's inner workings for the initialization phase in respect to its mood or personality (e.g. chaotic vs. methodic searches), but still maintaining the core execution (e.g. gardening task). This can result in more diversity of IVA's behaviors. The phases also allow tracking of the plan – e.g. the agent is only searching for items or the agent is already doing some cleanup. The overall chaotic and blackbox like behavior of a HRP plan is compensated in some degree, limiting it only to the execution phase. There is also the issue of separating processes that can be executed automatically (like object searches and cleanup) providing an "easier to design" concept.

The introduction of phases also allows adding of semantic information which can be evaluated during or before execution of the SFP. We consider the object relevancy as key information in respect to planning further activities and evaluating execution – the SFP might report fail, even before the execution starts, because it is aware of which objects it might need, and those are not available. The HRP concept is capable of such evaluation, but the plans might be too complex to design and maintain.

Object Relevancy

Problems of planning and deliberation require additional information introduced into the SFP to be able to perform some reasoning. We introduce the *Object Relevancy* (OR) into the SFP's structure. The idea behind OR is the observation, that most human and IVA activities are utilizing objects – the physical means to achieve goals. We introduce the *object list* into the SFP's semantic information set (i.e. specified for every plan by the AI designer). We provide a simple dining example to illustrate this.

(Fork[30 sec] & Knife[45 sec] & Spoon[60 sec]) || (Spoon[no limit] & Knife[600 sec]) || (Spoon[10 sec])

The semantic information is divided into sets of objects which at least one has to be satisfied – i.e. the IVA has to acquire or see these objects. The object list specification is also extended with a timeout information (per object), providing a guideline on how long the objects are to be searched for. Secondly, we introduce the *object classes* into the SFP. The object classes can be viewed as a description tag, describing properties and purpose of the objects. This concept was inspired by the academically unpublished game mechanics and structure for objects used in the game Sims©. The so-called *smart object* [*Champandard*, 2007] publish information to their surroundings – what *needs* of the in-game agents they satisfy [*Forbus*, 2002] – providing an *advertising concept*, allowing the agents to search for objects that suit them and their *needs*.

These two concepts provide necessary semantic information allowing us to perform some reasoning about the SFP and their purpose and their mutual relation.

Proof-of-concept prototype

We implemented a simple C++ based prototype, where we implemented all the above mentioned mechanisms [*Plch*, 2009]. The prototype is a simple 2D world home to various agents – a dog, a gardener, and a lubmerjack (Figure 3). We implemented 8 different scenarios to test the architecture in respect to the mentioned problems. For example, we tested the switching behavior with a scenario where the agent was a lumberjack/warrior cutting down trees to build a fort, but when an enemy disturbed him, he took out his shield and engaged him with his axe – the agent switched from woodcutter to warrior with a smooth transition. When utilizing HRP, the agent would put the axe in his bag (termination of "woodcutter") and take it out again (initialization of "warrior"). Our mechanism is smoother – the axe stays in hand. The deep preparation problem was tested by a scenario, where the lumberjack agent uses an axe which could get blunt, and acquires a sharpening tool when getting the axe, because it is required in the subplan ("sharpen axe") of the "woodcutter" plan used as his executed activity. This information is acquired during analyzing subplans during the initialization of the "woodcutter" plan.

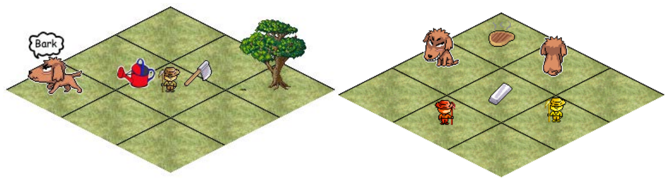

Figure 3. Prototype screenshot – dog and farmer agents are shown. Left image –dog is barking and farmer wants to water plants and cut down a tree on his field. Right image – two dogs are present, left dog is happy, right dog is ashamed. Two farmers are present, the left (red) one is angry the right (yellow) one is happy.

The scenarios ran in real-time, where the average AI engine's use of computational resources was negligible. The engine of the simulation took the majority of the computational resources, which was still tolerable (less than 10% CPU time and less then 100MB RAM). We used an Intel Pentium Dual Core system, with less than 2GB RAM operating a Linux operating system.

Future Work

The next step in our research is to create much more complex 3D testing environment utilizing a computer game (e.g. Half-Life 2 [*Valve*, 2004]) or a military simulation (e.g. Virtual Battlespace 2 [*Bohemia Interactive,* 2007]). We also want to further expand the SF-HRP concept into hybrid architecture – combining planning (e.g. Hierarchical Task Networks) and reactive paradigms.

Conclusion

We combined the FSM with the HRP concept introducing phases thus creating the SF HRP architecture capable of overcoming the presented issues of reactive planning. Our architecture still maintains HRP's core mechanisms (i.e. BE-Tree structure, the action selection algorithm, if-then rules). Therefore, we manage to retain the important trait of timely execution and introduce more control over the plan's execution by extending the RP into SFP. The SF-HRP's modularity provides a more adaptable design, which leads to more complex and believable behavior.

Our architecture was designed to overcome the observed issues of the HRP concept. The statefullness of SFPs provides basic capability to overcome transitional behavior issues by providing explicit phases the SFP manages when transitioning from one behavior to another. The ability to have specific transitions (besides generic transitions) based on the mutual relationship between SFPs provides a more adaptable architecture.

The additional information on per SFP Object Relevancy provides information use full for planning, deliberation, cleanup behavior as well as information of online/offline analysis (e.g. future need for objects). This information can be used for more complex reasoning about object acquisition and more sophisticated choices for plan alternatives to satisfy a goal.

As a part of our testing, we implemented a proof-of-concept implementation of a 2D environment inhabited by various agents. We tested 8 scenarios simulating mentioned problems, where HRP was not able to manage producing believable behavior in timely fashion.

Acknowledgments This work was partially supported by a student grant GA UK No. 0449/2010/A-INF/MFF, and by project P103/10/1287 (GA ČR).

References

Brom, C.: Hierarchical Reactive Planning: Where is its limit? In: Proceedings of MNAS workshop. Edinburgh, Scotland, (2005)

Bída, M., Brom, C., Popelová, M.: To Date or Not to Date? A Minimalist Affect-Modulated Control Architecture for Dating Virtual Characters. In: Vilhjálmsson, H.H.,Kopp, S., Marsella, S., Thórisson, K.R. (eds.) IVA 2011. LNCS, vol. 6895, pp. 419–425. Springer, Heidelberg (2011)

Bryson, J.: Intelligence by Design: Principles of Modularity and Coordination for Engineering Complex Adaptive Agents. PhD thesis, Massachusetts Institute of Technology, p59-75, (2001)

Champandard, A.J.: AI Game Development: Synthetic Creatures with Learning and Reactive Behaviours (Chapter 8), New Rider, (2003)

Chamapandard, A.J.: Living with the Sims' AI: 21 Tricks to Adopt for your game (2007), URL: http://aigamedev.com/reviews/the-sims-ai [5.5.2011]

Forbus, D.F.: Under the Hood of the Sims, CS 395 Game Design (2002) URL: http://www.cs.northwestern.edu/~forbus/c95-gd/lectures/The_Sims_Under_the_Hood_files/v3_document.htm [5.5.2011]

Ghallab M., Nau D., S., Traverso P.: Automated Planning: Theory and Practice, Elsevier, (2004)

Isla, D.: Handling Complexity in the Halo2, In: Games Developer Conference (2005)

Loyall, A., B.: Believable Agents, Ph.D. thesis, Tech report CMU-CS-97-123, Carnegie. Mellon University, (1997)

Mikula, T.: Hierarchical reactive planning with transitions, Bachelor Thesis Matematicko-fyzikální fakulta, Univerzita Karlova, Praha (2006)

Plch, T.: Action selection for an animat, Diploma thesis, Faculty of Mathematics and Physics, Charles University, Prague, (2009)

Plch, T., Brom, C.: Enhancements for reactive planning - tricks and hacks, In: Proceedings of SOFSEM (2010)

Riedl, M.O. and Stern, A.: Believable Agents and Intelligent Scenario Direction for Social and Cultural Leadership Training. In: Behavior Representation in Modeling and Simulation Conference (2006)

Russel, S., Norvig, P.: Artificial Intelligence: A Modern Approach, Prentice Hall, (2003)

Wooldridge, N.: AN Introduction into MultiAgent Systems, John & Wiley & Sons (2002)

Valve Software: Half Life 2, Valve Software (2004)

Bohemia Interactive Australia: Virtual Battlespace 2, Bohemia Interactive Studio (2007)