Constraints-Based Complex Behavior in Rich Environments

Jan M. Allbeck[1] and Hadas Kress-Gazit[2]

[1] Volgenau School of Information, Technology and Engineering, George Mason University, Fairfax, VA 22030
[2] Sibley School of Mechanical and Aerospace Engineering, Cornell University, Ithaca, NY 14853

Abstract. In order to create a system capable of planning complex, constraints-based behaviors for an agent operating in a rich environment, two complementary frameworks were integrated. Linear Temporal Logic mission planning generates controllers that are guaranteed to satisfy complex requirements that describe reactive and possibly infinite behaviors. However, enumerating all the relevant information as a finite set of Boolean propositions becomes intractable in complex environments. The PAR (Parameterized Action Representation) framework provides an abstraction layer where information about actions and the state of the world is maintained; however, its planning capabilities are limited. The integration described in this paper combines the strengths of these two frameworks and allows for the creation of complex virtual agent behavior that is appropriate to environmental context and adheres to specified constraints.

Keywords: Complex Behaviors, Representations, Agent Architectures.

1 Introduction

Attempting to instruct artificial entities, be they robots or agents in a virtual environment, requires representing information about the actions, environment, and agents and being able to efficiently process this data to interpret and execute the instructions. In this paper we describe the integration of two subsystems that form a framework for instructing agents to perform *complex behaviors* in *complex environments* while adhering to dictated *constraints*. Instructions may indicate what behaviors an agent should perform, but they can also provide constraints on how they should be performed. A constraint might completely prohibit an action (e.g. *Don't run*). Other constraints may impact the timing or priority of actions (e.g. *Do your homework before playing video games*). There could also be spatial constraints (e.g. *Stay out of Room 12*). Finally, constraints may include a variety of factors that form an overall context (e.g. *Do not go into a classroom when a class is in session*).

Instructed behaviors can also come in a variety of forms. There may be simple imperatives containing a single verbs (e.g. *Pickup the pencil*) or complex

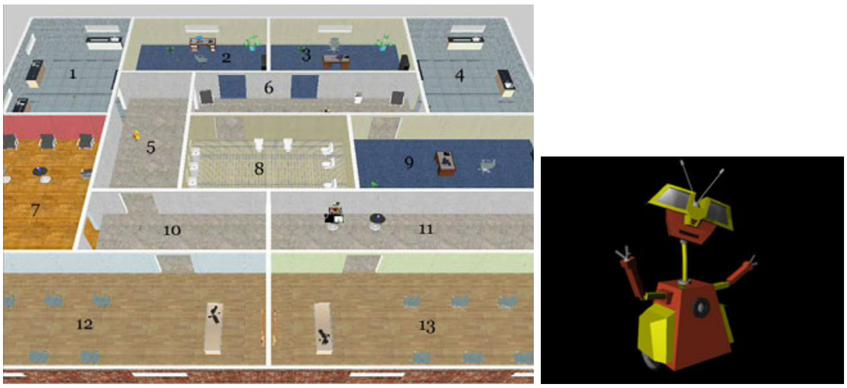

Fig. 1. (a) Virtual World with room numbers shown. (b) Murray, the virtual robot.

multi-step behaviors involving a sequence of actions or actions that should be performed in parallel (e.g. *While programming, drink coffee*). Other directives may require the agents to plan (e.g. *Search the building for weapons*). Finally instructions may include standing orders that dictate how the agents should handle scenarios they may encounter (e.g. *If you see a weapon, pick it up and move it to Room 13*). All of these behaviors can take place in a rich environment with many objects of many different types. Furthermore, these environments are dynamic. Other characters are operating within the environment, moving objects and changing their states and properties. Some instructions such as *Do not go into a classroom when a class is in session*, certainly address this dynamic nature.

In this paper, we present a framework that provides a strong representational and operational foundation for instructing virtual agents. The first component, the Parameterized Action Representation (PAR), provides a rich representation, a system for grounding the terms, and low level controllers for agents in a virtual environment. The second component, Linear Temporal Logic Mission and Motion Planning, is a framework for creating controllers such that the agent is guaranteed to satisfy high level task specifications, if they are feasible. The integration of these components provides constraints based complex planning and behaviors in rich environments. Additionally, we introduce the notion of meta-PARs. MetaPARs extend traditional, atomic PARs to provide more information about the desired behavior and constraints on it. MetaPARs aid the connection between PAR and the logic framework and will also provide a target representation for natural language parsers, semantic taggers, or pragmatic processors.

To help highlight the features of our framework and ground our discussions, we will present two example scenarios. One of the scenarios being examined involves using virtual robots in a building clearing exercise (See Figure 1). The robot searches the environment, obeying constraints, and reacting as instructed to the objects he discovers. As an example, Murray, the virtual robot, may be instructed: "*Search rooms 1, 2, 3 and 4. If you see a dead body, abandon the* *search and go to room 11. If you see a bomb, pick it up and take it to room 13 and then resume the search.*"

To further highlight advantages of the integration, we also present examples from another scenario. This scenario involves replacing old computers with new ones. It is set in a university environment and contains instructions such as, "*Replace all of the computers. Do not enter a classroom during a class. Make sure that classrooms always have at least one computer. Replace all classroom computers before office computers. Do not remove a computer that someone is using.*" Throughout the rest of this paper we will reference these two scenarios to frame examples and illustrate the advantages of our integration.

2 Parametrized Action Representation

At a finer level of detail, effective instructions may include, implicitly or explicitly, several additional elements. There are the core semantics that indicate the action or actions to be performed. There may even be an action structure composed of several sub-actions. There are also the participants of the action, including the agent being instructed to perform the action and any objects involved in the action (e.g. Pickup the *cup*). Instructions may also contain information related to the context, including path information, manner, or purpose. Initiation conditions are often implicit. Is the agent capable of performing the action? What state must the world and agent be in before the action is performed? Finally, effective instructions include either implicitly or explicitly termination conditions that indicate when an action has been completely successfully or when it cannot be. The Parameterized Action Representation (PAR) includes all of these elements and more [4].

In the PAR system, goal states are not specified and then planned for. Instead, agents are instructed to perform actions. These instructions can be sent from a high level controller [1] or from a human user [3]. The only planning included in the current PAR system is simple backward chaining done through explicitly pairing the preconditions of an action with other actions that can fulfill the conditions if required. The most common pairing is collocating the agent with an object participant of the action through locomotion. For example, if instructed to pickup an object, an agent will automatically move to the object. The location of the object is stored as a parameter of the object (initialized automatically from the environment and updated through action post-assertions). Animating the locomotion behavior and navigation is a function of the *Simulator* (See Figure 2) [12].

PAR is a rich representation that has been linked to motion generators in simulation frameworks [1,3,4]. PAR actually includes a representation of both actions and objects. Each are stored in hierarchies in a MySQL database called the *Actionary* (See Figure 2). The *Actionary* holds information about generic/uninstantiated actions and objects (e.g. what properties change as a consequence of an action or what properties are true of all chairs) as well as specific information about instantiated actions and objects (e.g. $Agent_4$ *Pickup*

Cup_0 or $Chair_6$ is *RED* and located in $Classroom_7$). Instantiated actions and objects are leaves in the hierarchies inheriting (or if desired overwriting) parameters from their uninstantiated parents. As actions execute, they change the state and properties of objects and agents in the simulated environment. These changes are also automatically registered in the instantiated objects in the *Actionary*, which then serves as a representational *World Model.*

Furthermore, the hierarchies provide a convenient classification system. An agent might be instructed to explore the environment and pickup all weapons and move them to a certain room. When the agent perceives an object in the environment, the agent's *Agent Process* (i.e. the PAR control process for the agent (See Figure 2)) checks to see if the perceived object is a descendant of the *Weapon* node in the object hierarchy and if so reacts accordingly. Hence instructions can use broader more general terms without explicitly enumerating predicates.

Instantiated objects also contain a lot of useful parameters including properties (e.g. color and age), postures (e.g. open or closed), states (e.g. on or off and free or in-use), and location among many others. Here *location* refers to another PAR object and not a three-dimensional coordinate, which is another parameter, namely *position.* For example, $Computer_8$ is located in $Office_2$ or Gun_9 is located in the contents of $Agent_4$. *Contents* is another PAR object parameter. It provides a list of all of the objects in a location. Many of these object parameters are updated automatically as post-assertions of actions. For example, *Pickup* Obj_0 will automatically set the *location* of Obj_0 as the agent executing the action and add Obj_0 to that agent's *contents.* In other words, successful execution of a *Pickup* action implies that the agent is now holding the object. Furthermore, the position of the object will automatically change with the position of the agent until an action, such as *Drop* is performed to release the relationship. Because all of these parameters are maintained, they can easily be referenced in conditions for modifying behaviors.

While PAR is a rich representation that includes the semantics of actions and objects and provides a level of abstraction, it is not a logic. Plan constraints such as *Do not enter* $Room_7$ and *Do not remove a computer that is in use*, cannot be represented by PARs. Similarly there is no convenient way to represent general sets or quantifiers. These limitations have been overcome through integration with a logic framework.

3 Linear Temporal Logic Mission and Motion Planning

The Linear Temporal Logic mission and motion planning framework [9,10] creates continuous control inputs for a robot such that its (possibly infinite) behavior satisfies a high-level complex specification that includes temporal operators and constraints. The idea at the heart of this approach is that a continuous control problem (i.e. finding velocity commands for the robot's movement/continuous actions) is abstracted into a discrete domain where boolean propositions correspond to basic robot states. For example, a proposition $Room_1$

will be true whenever the physical location of the robot or agent is contained in the area defined as Room 1. Then, the specification is written as temporal logic formulas and synthesis techniques generate a correct-by-construction automaton that when executed activates atomic controllers that drive the robot according to the specifications. In the following we describe the different aspects of this framework.

3.1 Specification

Formulas written in a fragment of Linear Temporal Logic (LTL) are used to specify the desired agent behavior and any information about the environment in which it is operating. These formulas have specific structure [10] and are defined over a set of boolean propositions which are task and agent specific; they describe the regions in the environment, events that may be detected (such as a bomb is seen or a class is in session) and actions the agent can perform (such as pick up, drop or replace computer).

In addition to the propositions, the formulas include boolean operators ($\neg$ 'not', $\vee$ 'or', $\wedge$ 'and', $\Rightarrow$ 'imply', etc.) and temporal operators ($\Box$ 'always', $\Diamond$ 'eventually', $\bigcirc$ 'next'). Loosely speaking, the truth of an LTL formula is defined over infinite executions of a finite state machine; $\Box p$ is true if p is true in every step of every execution, $\Diamond p$ is true if for every execution there is a step in which p becomes true and $\bigcirc p$ is true if on every execution, p is true in the next step. The formula $\Box\Diamond p$ is true if p becomes true infinitely often. We refer the reader to [6] for a formal description of LTL.

The LTL formulas describe several aspects of the agent's behavior: its motion constraints, for example $\Box(Room_1 \Rightarrow (\bigcirc Room_1 \vee \bigcirc Room_5))$ states that when the agent is in room 1, it can either stay there or move to an adjacent room in the next step. This part of the LTL formula is generated automatically from a given map of the environment. Other aspects are the desired reaction to events, for example $\Box(\bigcirc bomb \Rightarrow \bigcirc PickUp)$ encodes that the agent should pick up a bomb if it encounters one, and desired motion, for example $\Box\Diamond(Room_1 \vee Room_2)$ requires the agent to go to either room 1 or room 2.

3.2 Automaton Synthesis and Execution

Once the specification is written in the logic, the formula is synthesized into an automaton such that every execution of the automaton is guaranteed to satisfy the specification, if it is feasible. If the specification cannot be guaranteed, no automaton will be generated and the user gets an error message.

The (discrete) synthesized automaton is then transformed into a hybrid controller that provides continuous control commands to the agent. This controller, based on the state of the agent and its environment, governs the execution of atomic continuous controllers whose composition induces the intended robot behavior.

4 Integration of PAR and the LTL Framework

The strength of PARs lay in providing a rich description of agents, the actions they can perform and the world around them. However, while sequential planning can be performed by backward chaining, constraint-based, temporal, complex behaviors cannot be planned by the system. Furthermore, there is no built in mechanism for remembering events that occurred in the past that influence future behavior. Integration with the LTL planning framework provides a solution to these issues.

On the other hand, the LTL planning framework naturally creates complex behaviors that are guaranteed to satisfy different constraints and temporal relations, but because it plans over a finite set of Boolean propositions, all possible information has to be enumerated, which makes the approach intractable when the environment becomes too complex. Furthermore, knowledge of the agent state (for example, it is holding a bomb) has to be captured using extra propositions whose truth values need to be reasoned about. The PAR formalism provides the LTL planning framework an abstraction layer where information about possible actions and the state of the virtual agent is maintained by PAR and only information needed for the planning is passed along. Together these two subsystems complement each other and provide a foundation for specifying and executing constraints-based complex behaviors in rich environments.

Figure 2 provides an overview of the integrated system components and their connections. Before a simulation begins, an *External Controller* is responsible for defining a scenario and hence creating the environment, objects, and agents.

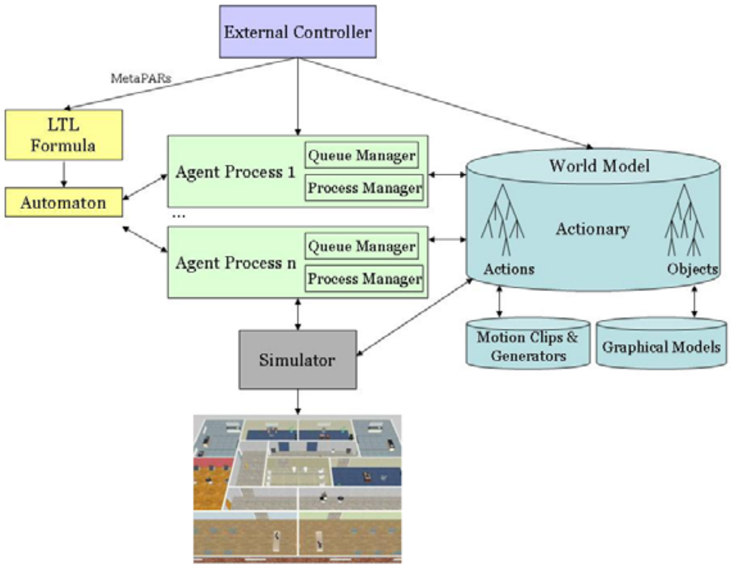

Fig. 2. Integrated System

From this information, LTL propositions are created. The information is also stored in the *Actionary* and a PAR *Agent Process* is created for each agent.

The *External Controller* also defines the behavior of the agents. The ultimate aim is to instruct virtual agents through natural language, in which case this external controller would include parsers, semantic taggers, and pragmatic processors. Currently we are using GUI forms. PARs have previously been linked to natural language processors [4] and the LTL framework receives input from structured English [9]. In order to gain the planning and reasoning capabilities of the LTL framework while maintaining the rich representation of the PAR formalism, the next section introduces the notion of *MetaPARs.* These special PARs include in addition to the standard PAR fields [4] special fields that provide information regarding desired behavior and constraints. Once these fields are defined they are converted, together with the workspace description, into LTL formulas that then get synthesized into an automaton that governs the execution of atomic PARs.

During a simulation, perceptions are passed from the *Simulator* to the *Agent Processes.* Transitions between states in the *Automaton* are based on these perceptions as well as other information stored in the *World Model* (i.e. *Actionary*). LTL propositions in the states then specify the atomic PAR actions that should be performed. These actions are processed by the *Agent Processes* resulting in updates to the *World Model* and control over the virtual agents displayed in the *Simulator.*

5 MetaPARs

MetaPARs are an integrated representation of PARs and LTL expressions. The added fields augment traditional PARs with information further detailing the desired behaviors and constraints. A PAR is considered a *MetaPAR* when it includes the following special fields:

- *Primary Action* - The main objective of the PAR. Includes behaviors such as search (area for object), explore (area), find (object), replace (area, old object, new object), goto (region), etc. These actions can be terminating or infinite.
- *Primary Action Parameters* - Parameters for the primary action such as description of the area to explore or the object to find.
- *Memory i ON Condition* - Condition for setting a memory proposition used to indicate that an event of interest occurred. For example, seeing a body needs to be remembered because it alters the future behavior of the agent.
- *Memory i OFF Condition* - Condition for reseting memory proposition i.
- *Priority i_j Condition* - Condition for behavior that is in priority i_j.
- *Priority i_j Reaction* - Behavior for the agent if the condition is true. We distinguish between two types of reactions; *liveness*, behaviors that have to eventually occur, and *safety*, behaviors the agent must always satisfy.
- *Priority i_j Condition Type* - Indicate the reaction is one of two types. For *If and only if* (iff) the action defined in the reaction field should only occur

if the condition is true. For *if* the action has to occur when the condition is true but can occur in other situations as well.

All the Condition and Reaction fields contain LTL formulas over the set of atomic propositions (atomic PARs and memory propositions). These formulas can contain, other than the Boolean operators, only the 'next' temporal operator $\bigcirc$.[1]

The priorities define which reaction takes precedence. Priority i_j is higher than k_l for $i < k$ and therefore reaction k_l will be executed only if condition i_j is not active. Reactions with priorities i_j and i_k have the same priority and the execution of one does not dependent on the status of the condition of the other. When defining these metaPARs the user needs to make sure reactions with the same priority do not contradict.

Each unique MetaPAR is automatically translated into a conjunction of LTL formulas corresponding to the MetaPAR's primary action, parameters, memory and reactions. These formulas are defined over the set of propositions that correspond to the environment of the agent, the objects in the environment, the memory propositions and a termination proposition (*done*) for primary actions that terminate. Table 1 describes LTL formulas that are generated from the primary action and parameters of several different MetaPARs. These initial parameters were created after extended discussions and preliminary corpus analysis from the robot building clearing domain. It is straight forward to extend this table to different actions as the need arises.

Table 2 describes the mapping between the rest of the MetaPAR fields (memory propositions, reactions) and their corresponding LTL formulas. There, propositions of the form M_i correspond to events that need to be remembered, that is, the proposition M_i becomes true when the 'ON' event is detected and false when the 'OFF' event is encountered. Propositions of the form C_i are used to represent conditions of required reactions. While not necessary, they simplify the formulas capturing the reactions and priorities. Note that some formulas, such as the ones defining the initial conditions of the propositions, are omitted here to maintain clarity.

Going back to our building clearing example, the MetaPAR *ExploreOnce* with parameters *Location* $= \{r_1, r_2, r_3, r_4\}$ contains one memory proposition (M_1) for remembering that *Dead* was true at some point in the past and four reactions. This MetaPAR translates to the following LTL formula where lines 4, 5 and 16 correspond to the primary action, 6 to the memory proposition and 7-15 to the reactions. Initial conditions of the propositions and the topology of the environment are captured in lines 1-3. The conditions of the reactions are represented using $C_1, C_{21}, C_{22}, C_{23}$. For example, C_1 becomes true when a dead person is encountered (line 7) and it is the condition for requiring the robot to go to room 11 and stay there (lines 8,9).

[1] For a formal description of the structure the reader is referred to [10].

Table 1. Translation of MetaPAR primary actions into corresponding LTL formulas

MetaPAR Primary Action	**Parameters**	**Corresponding LTL Formula**
Search until finding	Location $L = \{l_1, l_2, \dots\}$, Object *obj*	$\wedge_{l\in L}\neg srch_l \wedge \neg done$ {Didn't search and not done} $\bigwedge_{l\in L}\Box((l \vee Srch_l) \Leftrightarrow \bigcirc Srch_l)$ {If you are in l or you searched there before, then remember you already searched l } $\wedge\Box(\bigcirc obj \Rightarrow \bigwedge_i (r_i \Leftrightarrow \bigcirc r_i))$ {If you see the object, stay where you are} $\wedge\Box(\bigcirc obj \Leftrightarrow \bigcirc done)$ {If you see the object you are done} $\wedge\Box\Diamond((\bigwedge_{l\in L} Srch_i) \vee done \bigvee_{C\in condition} C)$ {If has to be true that infinitely often you either searched the locations, are done or a condition is true}
Explore once	Location $L = \{l_1, l_2, \dots\}$	$\wedge_{l\in L}\neg srch_l \wedge \neg done$ {Didn't search and not done} $\bigwedge_{l\in L}\Box((l \vee Srch_l) \Leftrightarrow \bigcirc Srch_l)$ {If you are in l or you searched there before, then remember you already searched l } $\wedge\Box((\bigwedge_{l\in L} \bigcirc Srch_i) \Leftrightarrow \bigcirc done)$ {If you searched all locations, you are done} $\wedge\Box\Diamond(done \bigvee_{C\in condition} C)$ {If has to be true that infinitely often you either are done or a condition is true}
Explore infinitely	Location $L = \{l_1, l_2, \dots\}$	$\bigwedge_{l\in L}\Box\Diamond(l \bigvee_{C\in condition} C)$ {Infinitely often go to l unless a condition is true}
Find	Object *obj*	$\bigwedge_i \Box\Diamond(r_i \vee obj \bigvee_{C\in condition} C)$ {Infinitely often go to all regions unless you find the object or a condition is true}
Goto	location formula $\mathcal{L}$	$\neg done$ {Not done} $\wedge\Box(\bigcirc\mathcal{L} \Leftrightarrow \bigcirc done)$ {If you are in $\mathcal{L}$ you are done} $\Box\Diamond(done \bigvee_{C\in condition} C)$ {Infinitely often done unless a condition is true}

Table 2. Mapping between the MetaPAR fields and the corresponding LTL formulas

MetaPAR Fields	**Corresponding LTL Formula**
Memory i *ON Condition* (once ON stays ON)	$\Box((ON\ Condition \vee M_i) \Leftrightarrow \bigcirc M_i)$
Memory i *ON Condition*, *OFF Condition*	$\Box(((ON\ Condition \vee M_i) \wedge \neg OFF\ Condition) \Leftrightarrow \bigcirc M_i)$
Priority i_j *Condition*	$\Box(Condition \Leftrightarrow \bigcirc C_{ij})$
Priority i_j *Reaction* (safety - iff)	$\Box((\bigcirc C_{ij} \wedge \neg \bigvee_{k<i,\forall l} \bigcirc C_{kl}) \Leftrightarrow Reaction)$
Priority i_j *Reaction* (safety - if)	$\Box((\bigcirc C_{ij} \wedge \neg \bigvee_{k<i,\forall l} \bigcirc C_{kl}) \Rightarrow Reaction)$
Priority i_j *Reaction* (liveness - iff)	$\Box\Diamond((C_{ij} \wedge \neg \bigvee_{k<i,\forall l} C_{kl}) \Leftrightarrow Reaction)$
Priority i_j *Reaction* (liveness - if)	$\Box\Diamond((C_{ij} \wedge \neg \bigvee_{k<i,\forall l} C_{kl}) \Rightarrow Reaction)$

$$\bigwedge \neg M_1 \wedge \neg Pick \wedge \neg Drop \wedge \neg C_1 \tag{1}$$

$$\bigwedge_{i\in\{1,\ldots,4\}} \neg Srch_i \wedge \neg C_{21} \wedge \neg C_{22} \wedge \neg C_{23} \wedge \neg done \tag{2}$$

$$\bigwedge \text{Topology of the environment} \tag{3}$$

$$\bigwedge_{i\in\{1,\ldots,4\}} \Box((r_i \vee Srch_i) \Leftrightarrow \bigcirc Srch_i) \tag{4}$$

$$\bigwedge \Box((\bigwedge_{l\in L} \bigcirc Srch_i) \Leftrightarrow \bigcirc done) \tag{5}$$

$$\bigwedge \Box((\bigcirc Dead \vee M_1) \Leftrightarrow \bigcirc M_1) \tag{6}$$

$$\bigwedge \Box(\bigcirc M_1 \Leftrightarrow \bigcirc C_1) \tag{7}$$

$$\bigwedge \Box\Diamond(C_1 \Rightarrow r_{11}) \tag{8}$$

$$\bigwedge \Box((\bigcirc C_1 \wedge r_{11}) \Rightarrow \bigcirc r_{11}) \tag{9}$$

$$\bigwedge \Box((\bigcirc seeBomb \wedge \neg \bigcirc haveBomb) \Leftrightarrow \bigcirc C_{21}) \tag{10}$$

$$\bigwedge \Box((\bigcirc C_{21} \wedge \neg \bigcirc C_1) \Leftrightarrow \bigcirc Pick) \tag{11}$$

$$\bigwedge \Box((\bigcirc haveBomb \wedge r_{13}) \Leftrightarrow \bigcirc C_{22}) \tag{12}$$

$$\bigwedge \Box((\bigcirc C_{22} \wedge \neg \bigcirc C_1) \Leftrightarrow \bigcirc Drop) \tag{13}$$

$$\bigwedge \Box(\bigcirc haveBomb \Leftrightarrow \bigcirc C_{23}) \tag{14}$$

$$\bigwedge \Box\Diamond((C_{23} \wedge \neg C_1) \Rightarrow r_{13}) \tag{15}$$

$$\bigwedge \Box\Diamond(done \vee C_1 \vee C_{21} \vee C_{22} \vee C_{23}) \tag{16}$$

The LTL formula for the clearing example was automatically synthesized into an automaton containing 582 states. Figure 3 depicts part of the automaton; the circles (states) contain the action propositions that are true in that state.

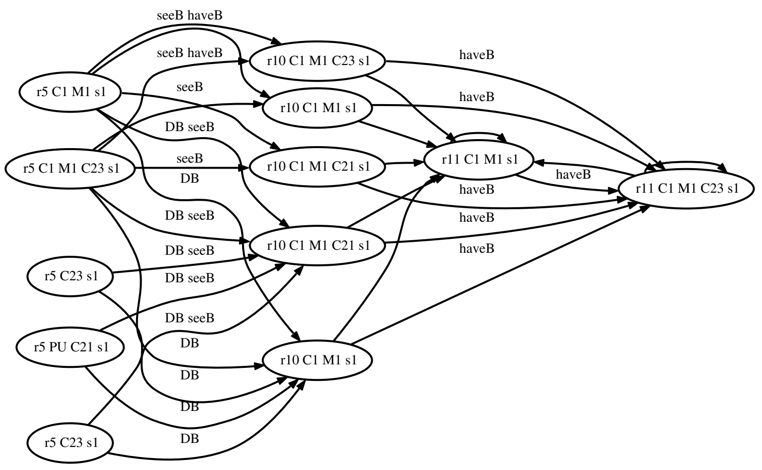

Fig. 3. Part of the automatically synthesized automaton for the clearing example. A possible execution could be starting from the top left most state: the agent is in room 5 and has searched room 1 (s_1 is true) and seen a dead body before (M_1 is true). Then, it goes to room 10. He ignores (does not pickup) bombs he sees along the way since the reaction to dead bodies has priority over bombs and then he reaches room 11, as was required.

These propositions relate to the behavior of the agent, for example, PU refers to the agent picking up a bomb. The arrows (transitions) are labeled with sensor propositions that must be true for that transition to take place, for example, if a dead body (DB) was found in room 5, the agent must go to room 10 next. These propositions are easily evaluated by PAR system. All transitions to and from other states in the automaton were omitted for clarity.

Interestingly, while executing this metaPAR Murray exhibited an unwanted, while correct, behavior. It would pick up a bomb, take it to Room 13, drop it and then begin an infinite loop of picking up the bomb and dropping it. This behavior was due to an incomplete specification; Murray was never instructed to ignore bombs in Room 13. This behavior was easily fixed by inserting a condition '$\bigcirc seeBomb \wedge r_{13}$' with reaction '$\neg \bigcirc PU$' and with priority 2 and shifting down the rest of the conditions. Figure 4 depicts snapshots of a simulation of Murray clearing a building.

Our second scenario, replacing old computers with new ones (See Figure 5), further highlights some of the capabilities of this integration. For example constraints, and therefore behaviors, can be based on dynamic conditions. Naturally, we can specify a constraint such as *Do not go into Room 12*, but the framework can also handle constraints such as *Do not enter a classroom during a class*. The applicability of this constraint changes as the simulation progresses. It is also possible to have ordering constraints, such as *Replace all classroom computers before office computers*, without strictly ordering all of the replace actions.

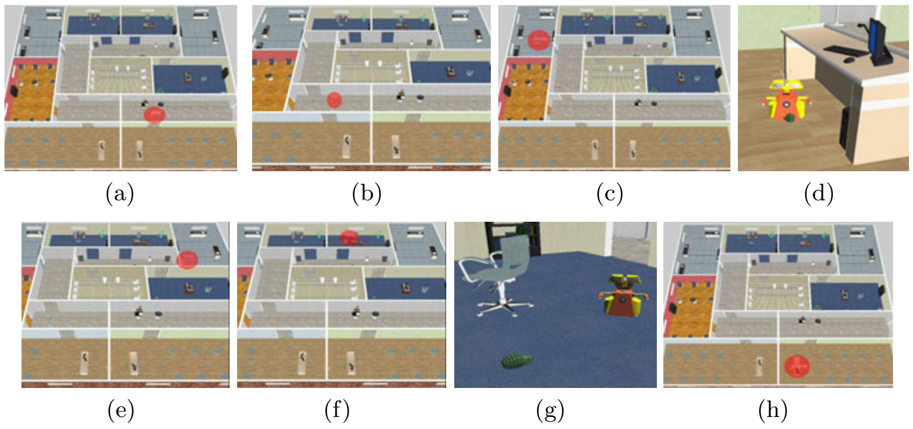

Fig. 4. Selected frames from the building clearing scenario (a) Murray begins in Room 11 and starts moving toward Room 1. (b) Murray passes through Room 10. (c) After passing through Room 5, Murray arrives at Room 1, sees a bomb, and picks it up. (d) He makes his way to Room 13 and drops off the bomb. (e) He then resumes the search and picks up a bomb in Room 4. (f) After dropping the bomb from Room 4 in Room 13, Murray picks up another bomb in Room 3. (g) Finally, he finishes his search in Room 2 where he picks up another bomb. (h) Murray drops off the bomb in Room 13 and the scenario ends.

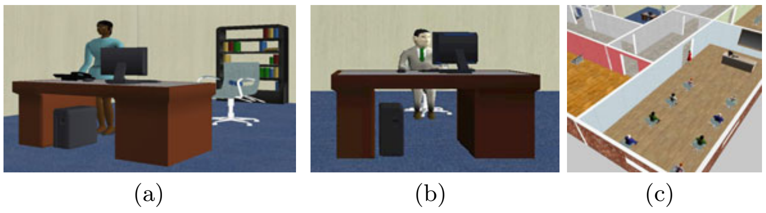

Fig. 5. Selected frames from the replacing computers scenario. (a) Replacing an office computer. (b) Office computer in use. (c) Class being held in a classroom.

6 Conclusion

While previous work on creating intelligent virtual agents (IVA's) has produced some interesting and sometimes complex behaviors [7,5,14,11], the work either did not allow users to input instructions to the characters or required them to tediously create decision and knowledge structures a priori. Many other efforts including [13] have focused on dialog planning which is beyond the scope of our current work. Like [8] and other work on IVA's, [13] also includes an element of plan recognition. The IVA's analyze user actions and recommend corrections or future actions based on stored plans for the given task domain. The focus of our framework is fulfilling complex instructions given by a user. As discussed in [2], it is important for planning to occur continuously and not be *baked* into the IVA. The LTL framework naturally provides plan flexibility since once the user changes a MetaPAR field a new plan is synthesized automatically.

The work presented in this paper was conducted as part of a large project aimed at answering the question of how robots (physical or virtual) can be instructed using natural language. In collaboration with linguists and natural language processing researchers, we have integrated the rich representation of PAR with the planning and reasoning capabilities of the LTL-based framework to create the underlaying representation and reasoning mechanism that will interface with the pragmatics and semantics of the language on one side and with the low-level robot control on the other. As a part of this project, researchers have begun collecting corpora that are being used to analyze the type of instructions that will be found in this domain so that components such as MetaPARs will be built with enough depth and robustness to meet the needs of the domain. As we learn more from these corpora, we will extend the MetaPAR representation. The ultimate analysis of the integration we have presented will come when all of the components of the entire system are joined together from the natural language parser through to a robot.

The resulting system enables complex, constrained-based behaviors for robots or virtual agents that are not possible in either framework alone. We have demonstrated the representations and frameworks through two rather different scenarios. There are, however, still limitations. We are currently using the LTL framework to determine the route an agent takes from one room to another. While the LTL framework will guarantee a successful route if one exists, the route generated is often suboptimal. Using way-point navigation, the PAR framework along with the *Simulator* does generate an optimal path. Unfortunately, constraints are not taken into account. We plan to address this by having the PAR framework generate a path that is then checked by the LTL framework to ensure that it does adhere to all constraints and when possible gives an indication of the spaces that should be avoided so that an alternative path can be obtained. As we further explore these domains and others, we may encounter other suboptimal plans that will lead to unnatural behaviors. We hope to mitigate these occurrences through the use of pragmatics and the existing semantics already available in PARs.

Furthermore, we have been testing this system on a single agent. In the future, we would like to construct teams of agents. We anticipate resource management issues as well as possible goal conflicts. The *Simulator* does include a resource manager that with extensions should handle some of these issues. Including agents statuses and referencing action priorities should help resolve some of the goal conflicts by determining which agent should take precedence, and then replanning for the others.

Acknowledgements

Partial support for this effort is gratefully acknowledged from the U.S. Army SUBTLE MURI W911NF-07-1-0216. We also appreciate donations from Autodesk and nVidia.

References

1. Allbeck, J.M.: Creating 3D Animated Human Behaviors for Virtual Worlds. Ph.D. thesis, University of Pennsylvania (2009)
2. Avradinis, N., Panayiotopoulos, T., Aylett, R.: Continuous planning for virtual environments. In: Vlahavas, I., Vrakas, D. (eds.) Intelligent Techniques for Planning, pp. 162–193 (2005)
3. Badler, N., Erignac, C., Liu, Y.: Virtual humans for validating maintenance procedures. Communications of the ACM 45(7), 56–63 (2002)
4. Bindiganavale, R., Schuler, W., Allbeck, J., Badler, N., Joshi, A., Palmer, M.: Dynamically altering agent behaviors using natural language instructions. In: Autonomous Agents, pp. 293–300. AAAI, Menlo Park (2000)
5. Cavazza, M., Charles, F., Mead, S.J.: Planning characters' behaviour in interactive storytelling. The Journal of Visualization and Computer Animation 13(2), 121–131 (2002), 10.1002/vis.285
6. Emerson, E.A.: Temporal and modal logic. In: Handbook of theoretical computer science. Formal Models and Semantics, vol. B, pp. 995–1072. MIT Press, Cambridge (1990)
7. Funge, J., Tu, X., Terzopoulos, D.: Cognitive modeling knowledge, reasoning and planning for intelligent character. In: Proceedings of ACM SIGGRAPH, pp. 29–38 (1999)
8. Johnson, W.L., Rickel, J.: Steve: An animated pedagogical agent for procedural training in virtual environments. ACM SIGART Bullentin 8(1-4), 16–21 (1997)
9. Kress-Gazit, H., Fainekos, G.E., Pappas, G.J.: Translating structured english to robot controllers. Advanced Robotics Special Issue on Selected Papers from IROS 2007 22(12), 1343–1359 (2008)
10. Kress-Gazit, H., Fainekos, G.E., Pappas, G.J.: Temporal logic based reactive mission and motion planning. IEEE Transactions on Robotics 25(6), 1370–1381 (2009)
11. Paris, S., Donikian, S.: Activity-driven populace: a cognitive approach to crowd simulation. IEEE Computer. Graphics and Applications 29(4), 34–43 (2009), 1669315
12. Pelechano, N., Allbeck, J.M., Badler, N.I.: Controlling individual agents in high-density crowd simulation. In: ACM SIGGRAPH / Eurographics Symposium on Computer Animation (SCA). ACM Press, San Diego (2007)
13. Smith, C., Cavazza, M., Charlton, D., Zhang, L., Turumen, M., Hakulinen, J.: Integrating planning and dialgue in a lifestyle agent. In: Prendinger, H., Lester, J.C., Ishizuka, M. (eds.) IVA 2008. LNCS (LNAI), vol. 5208, pp. 146–153. Springer, Heidelberg (2008)
14. Yu, Q., Terzopoulos, D.: A decision network framework for the behavioral animation of virtual humans. In: Proceedings of ACM SIGGRAPH/Eurographics symposium on Computer animation, pp. 119–128. Eurographics Association, San Diego (2007)