Intelligent Biohazard Training Based on Real-Time Task Recognition

HELMUT PRENDINGER and NAHUM ALVAREZ, National Institute of Informatics
ANTONIO SANCHEZ-RUIZ, Universidad Complutense de Madrid
MARC CAVAZZA, School of Engineering and Digital Arts, University of Kent
JOÃO CATARINO, JOÃO OLIVEIRA, and RUI PRADA, INESC-ID and Instituto Superior Técnico, Universidade de Lisboa
SHUJI FUJIMOTO, Faculty of Medical Sciences, Kyushu University
MIKA SHIGEMATSU, National Institute of Infectious Diseases

Virtual environments offer an ideal setting to develop intelligent training applications. Yet, their ability to support complex procedures depends on the appropriate integration of knowledge-based techniques and natural interaction. In this article, we describe the implementation of an intelligent rehearsal system for biohazard laboratory procedures, based on the real-time instantiation of task models from the trainee's actions. A virtual biohazard laboratory has been recreated using the Unity3D engine, in which users interact with laboratory objects using keyboard/mouse input or hand gestures through a Kinect device. Realistic behavior for objects is supported by the implementation of a relevant subset of common sense and physics knowledge. User interaction with objects leads to the recognition of specific actions, which are used to progressively instantiate a task-based representation of biohazard procedures. The dynamics of this instantiation process supports trainee evaluation as well as real-time assistance. This system is designed primarily as a rehearsal system providing real-time advice and supporting user performance evaluation. We provide detailed examples illustrating error detection and recovery, and results from on-site testing with students from the Faculty of Medical Sciences at Kyushu University. In the study, we investigate the usability aspect by comparing interaction with mouse and Kinect devices and the effect of real-time task recognition on recovery time after user mistakes.

Categories and Subject Descriptors: H.1.2. [**User/Machine Systems**]; H.5.1. [**Information Interfaces and Presentation**]: Multimedia Information Systems

General Terms: Human Factors, Experimentation

Additional Key Words and Phrases: Bio-safety risk management, training application, virtual worlds

---

The reviewing of this article was managed by associate editor Anthony Jameson.
This work is supported by the National Institute of Infectious Diseases. This work was supported partly by (1) the Health and Labour Science Research Grants, Research on Emerging and Reemerging Infectious Diseases (research on development of teaching materials and methodology to evaluate performance to strengthen bio-risk management [H20-Shinko-Ippan-009]) from the Ministry of Health, Labour and Welfare of Japan as a contract research; (2) the "Global Lab" Grand Challenge grant from the National Institute of Informatics; (3) Fundação para a Ciência e a Tecnologia, under project PEst-OE/EEI/LA0021/2013; and (4) the Spanish Ministry of Economy and Competitiveness under grant TIN2009-13692-C03-03.
Authors' addresses: H. Prendinger and N. Alvarez, National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan; emails: helmut@nii.ac.jp, nahum.alvarez.ayerza@gmail.com; J. Catarino, J. Oliveira, and R. Prada, INESC-ID, Avenida Professor Cavaco Silva, Taguspark - Edif'icio IST, 2744-016 Porto Salvo, Portugal; emails: {jcbpc, joao.delgado.oliveira, rui.prada}@tecnico.ulisboa.pt; A. Sanchez-Ruiz, Universidad Complutense de Madrid, 28040 Madrid, Spain; email: antonio.sanchez@fdi.ucm.es; M. Cavazza, School of Engineering and Digital Arts University of Kent, Canterbury CT2 7NT, UK; email: M.O.Cavazza@kent.ac.uk; S. Fujimoto, Faculty of Medical Sciences, Kyushu University, 3-1-1 Maidashi, Higashi-ku, Fukuoka 812-8582, Japan; email: shuuji@hs.med.kyushu-u.ac.jp; M. Shigematsu, National Institute of Infectious Diseases, Toyama 1-23-1, Shinjuku-ku, Tokyo 162-8640, Japan; email: mikas@nih.go.jp.

ACM 2160-6455/2016/09-ART21 $15.00
DOI: http://dx.doi.org/10.1145/2883617

ACM Reference Format:
Helmut Prendinger, Nahum Alvarez, Antonio Sanchez-Ruiz, Marc Cavazza, João Catarino, João Oliveira, Rui Prada, Shuji Fujimoto, and Mika Shigematsu. 2016. Intelligent biohazard training based on real-time task recognition. ACM Trans. Interact. Intell. Syst. 6, 3, Article 21 (September 2016), 32 pages.
DOI: http://dx.doi.org/10.1145/2883617

1. INTRODUCTION

Intelligent training applications have gained considerable importance in recent years, as they can help people to learn to handle situations in which on-site training is dangerous or impractical. Here, 3D environments offer an ideal setting to develop training applications because of their ability to convey a greater sense of realism than textbooks or 2D interfaces [Stocker et al. 2011]. This sense of realism is generally attained through accurate visual representation and physical behavior of objects. Furthermore, the success of training applications depends on the correct integration of knowledge-based techniques and natural interaction. Knowledge-based techniques are used to support the training procedures that users have to execute. Natural interaction, on the other hand, is used to support execution of those training procedures in a way that mimics real-world execution as close as possible. However, the more complex the training scenario is, the more effort is required in implementing both knowledge and interaction aspects in a way that delivers an accurate and realistic training experience.

Recently, complex training scenarios involving the handling and management of pathogens have received great attention. Medical educational institutes are considering bio-risk training as a requirement for medical researchers to strengthen their response capability to bio-terror crises [Gaudioso et al. 2009; Rao 2011]. However, real-world laboratory practice is impractical because of its extremely high risk and expense. Thus, such scenarios are good candidates to evaluate the potential of combining a knowledge-based procedure for training and a natural interaction paradigm.

In this article, we describe the implementation of an intelligent application for biohazard training. The application emulates a scenario in which users must handle a critical biohazard situation involving the spill of a contaminated blood sample from a broken bottle. Our work has been tested in the field, that is, with students from the medical sciences who are the target users of the system. Our vision is to include our training method in the official curriculum of university students. The primary contribution of our work to intelligent interactive systems is to demonstrate the real-time integration of a task model in a virtual environment, supporting task recognition from user interaction with 3D objects. The secondary contribution is the integration of gesture-based interaction and the investigation of its benefits.

The rest of the article is organized as follows. Section 2 describes related work on intelligent training systems, activity recognition, and gesture-based interfaces. Section 3 presents the system architecture and its components for natural gesture interaction, common sense reasoning, and task recognition. Section 4 explains how user gestures are interpreted within the task tree that encodes the biohazard protocol to provide real-time feedback to the users. Section 5 reports on our study, in which medical students used our training system. First, Kinect and mouse versions of our system are evaluated in terms of usability. Second, a "dynamic" advice version based on real-time task recognition is compared to a "static" advice version, in which users can ask for text explaining the procedure. Section 6 summarizes the article and presents our conclusions.

2. RELATED WORK

2.1. Virtual Training Systems

There exist many works on training systems integrating interaction techniques with knowledge-based methods. They share the goal of improving the efficiency of knowledge

transfer in situations in which text and pictures are insufficient for training, and the conviction of the adequacy of the approach to better learning procedural tasks. As we do in this article, some authors present systems with intelligent object behavior and manipulation capabilities. For instance, Angelov and Styczynski [2007] present a virtual 3D simulator that aims to teach about the maintenance of a power plant. However, the simulator lacks flexibility in that it allows only fixed steps to complete the scenario. Belloc et al. [2012] present a generic system for scenario description, intelligent objects, and protocols. Similar to our system, the application is separated into three layers: graphics and physics, objects and behaviors, and plans. They present two examples: the assembly of a hydroelectric generation unit and the assembly of a combustion engine. Once the objects are described, the training scenario is fairly simple; the user has to perform a series of correct steps and is able to proceed only when previous steps are completed.

A style of training by doing tests appears frequently in research. For instance, van Wyk and de Villiers [2009] show an application designed to teach accident prevention in a mine in which users have to select the correct answers at the right moment, and Corato et al. [2012] describe a computer vision application for detecting whether the protocol for washing hands before an operation in an operating room is followed. It uses augmented reality technology but does not need markers for detecting actions. Similar to our work, the desired behavior is contained in health protocols. A limitation of the presented work is that, even if the system recognizes the stages in the protocol, it does not have a provision for responding to user mistakes. In other words, that approach does not provide student monitoring or dynamic help or assistance.

There exist several works with virtual tutors, or online assistants. Johnson and Rickel [1997] describe Steve, a virtual tutor system for teaching air compressor functions. A planner is used for executing scripted exercises and for explaining the rationale of the tutor's actions, but the system does not provide information about user mistakes. If the user makes a mistake, the tutor will correct the user telling the next step.

Including a virtual tutor is not incompatible with other techniques, such as storytelling. Carpenter et al. [2006] describe a virtual training system that teaches students how to manage the communication of crisis-related information. The system's knowledge is stored as an event tree that works as a storyboard with different choices. The users can make decisions that have consequences in the scenario, and use 3D vision glasses and six screens to facilitate immersion. Cavazza and Simo [2003] use a qualitative simulation model to train medical students to respond to patients with circulatory shock states. Like ours, this approach integrates knowledge-based techniques with a 3D environment in the health domain, but focused on the virtual patient rather than interacting with the environment.

Similar to Gordon [2003], we use a graph containing the description of the training scenario. That work describes a protocol of identifying and constructing decision structures in virtual environment scenarios. The result of this process is a decision graph divided into stages: training scenarios are constructed like storyboards, and branching trees are used to simulate training decisions. While our representation shares some similarities, training in that work is carried out by selecting text options in a web page, rather than by interacting with a complex 3D environment. Further, Vidani and Chittaro [2009] propose a task model (called a concur task tree) to train emergency procedures that may assist disabled people. The model aims to be a complete representation, including time relations between events and task difficulty management, but it is unclear to what extent features of the system are implemented. Also, the work does not separate difference knowledge layers, such as task definition and common sense knowledge base, which limits extensibility.

There is other important work in the intelligent tutoring system community [Conati et al. 1997; Corbett et al. 2000; Vanlehn et al. 2005]. However, there is an important difference in that we are recognizing how a protocol is being followed rather than how a cognitive strategy is being applied. The main differences are: (1) the task representation is explicit, which makes the issue of partial ordering [Conati et al. 1997; Geib and Goldman 2009] less relevant; (2) we need to recognize the level of task completion rather than user intent or user strategy, which simplifies the recognition problem and does not require an inflation of node types corresponding to different elements of the strategy and its application, as in Conati et al. [1997]; and (3) the connection between user actions and task representations is simplified because of the explicit representation and the virtual reality context [Sukthankar et al. 2014].

2.2. Plan Recognition

Real-time interactive systems with semantic representation of events and plans need to be decoupled from the internals of the simulation engine; otherwise, the semantic layer would be a static specification working in only one domain for some concrete cases. To achieve this goal, researchers have developed several solutions, generally based on the idea of using different layers to represent different types of knowledge (graphical, physical, common sense, actions and goals, and so forth) and some communication mechanism among them. Although there exist several general-purpose architectures, none of them is a standard, and each system usually defines its own. For example, Latoschik and Fröhlich [2007] present a solid architecture for interactive systems based on the concept of semantic reflection that uses monitors, filters, and subscription lists for decoupling all possible systems that can participate in a virtual environment. However, the architecture does not specify a concrete semantic module or layer: each module can equally access the system's knowledge base. It is generic to the extent that implementing such a layer is possible, which is left to a future developer. In further research, Wiebusch and Latoschik [2012] present an architecture targeting the use of common sense knowledge and grounding, as well as using a message dispatching system. It includes the Web Ontology Language (OWL) ontology as a central knowledge base to unify access to information from different modules. Our system also decouples different types of knowledge in semantic modules. In particular, we developed a common sense database and a task recognition module, but communication is performed module to module instead of using a centralized database. Our system is also more focused on task representation since task recognition is a key feature in our system.

Taking into account common sense knowledge and information on the physical state of the environment, Schneider [2010] proposes a plan recognition approach based on the user's plan selection and actions that are represented as a probabilistic automaton. This automaton is converted to a dynamic Bayesian network, allowing the plan recognition process to occur at runtime. This approach is similar to ours because it also applies real-time monitoring, trying to infer the intended plan by observing the user actions and the states of the environment. It also tries to predict the user's probable next actions so that it can proactively provide additional instruction on how to correctly execute the chosen plan. Our approach, on the other hand, relies on a hierarchical representation of tasks such that complex goals can be decomposed into simpler ones. This representation is more intuitive for the experts in our domain than a flat representation, because security procedures are usually represented this way. It also allows the provision of feedback to the user using abstract description of goals instead of concrete actions. Finally, we do not use probabilities to compute the next most probable task to complete; we use a simple heuristic based on the hierarchical nature of our plans that seems sufficient for our purposes. The use of Dynamic Bayesian Networks to achieve intention recognition

can also be applied for different scenarios. In Tahboub [2006], the probabilistic intention inference is achieved by modifying the intention-action-state scenario and modeling it by Dynamic Bayesian Networks.

In general, plan-recognition algorithms require as inputs a sequence of the observed agent's actions and some type of model describing its acting capabilities and/or decision-making processes. Plan library–based approaches [Kautz and Allen 1986] use a collection of plans or recipes describing all possible sequences of actions that the agent can perform to select the most probable plan at a given time. This library of plans can be provided as Hierarchical Task Networks (HTNs) or partially ordered multiset context-free grammars [Geib and Goldman 2009; Kabanza et al. 2013], probabilistic grammars [Pynadath and Wellman 2000], Bayesian networks [Synnaeve and Bessière 2011], hidden Markov models [Bui et al. 2011], or Markov logic [Song et al. 2013]. Unlike plan-library–based approaches, inverse planning–based approaches [Ramirez and Geffner 2009, 2010; Baker et al. 2009] require only a description of the actions that the agent can execute. The intuition is that the most likely goal is the one for which the optimal plan is most consistent with the observations so far. Unfortunately, these approaches have a high overhead because they require invocation of a planner for each goal and observed action.

We have chosen to represent biohazard training procedures in our simulator using a hierarchical task model. These hierarchical representations are compact and intuitive, alleviating the work of the domain experts that were involved in the definition of the laboratory procedures. This way, HTN-based approaches are very related to our work. Geib and Goldman [2009] developed their approach to address difficulties in plan recognition listed as "the execution of multiple interleaved root goals, partially ordered plans, and failing to observe actions." However, these difficulties do not apply to our context due to the explicit task representation, direct connection between leaf nodes and actions, and direct access to actions in the virtual world. We face a simplified problem, closer to early plan-recognition methods, in which, for instance, the issue of multiple goals likelihood [Sukthankar et al. 2014] is not salient. The primary contribution of our work to intelligent interactive systems is to demonstrate the real-time integration of a task model in a virtual environment, supporting task recognition from user interaction with 3D objects. The DOPLAR algorithm [Kabanza et al. 2013] addresses the problem of computing the probability of a goal given the observations to date without having to enumerate all possible hypotheses. It uses a heuristic-weighted, model-counting algorithm that limits the number of generated plan-execution models by computing lower- and upper-bound likelihoods. The creation of these HTNs is, in general, a complex and costly task; thus, some authors have proposed different methods to alleviate the process. Bisson et al. [2015], for example, propose using a recursive neural network to automatically learn accurate HTN decision-making models of the observed agent.

Unfortunately, the integration of these advanced plan-recognition algorithms as a subsystem of a bigger architecture is usually problematic because either their implementation is not publicly available or it has important limitations (such as working only with propositional plans) that prevent its use in real applications. In addition, the inherent computational complexity of the plan-recognition problem makes it difficult to program solutions that work in real time. For these reasons, most of the e-learning and virtual training systems implement some *ad hoc* solutions to recognize the user's current goals, solutions limited in several aspects but enough for the purposes of the system.

In the literature, we can also find more specific systems designed for e-learning and training purposes and using plan-recognition techniques. Franklin et al. [2002] propose a complete system designed to assist the user in real time by recognizing *activities* that are sequences of processes represented using finite state automata. The system

contains features similar to ours, such as backtracking and real-time monitoring, but it neither uses hierarchical plans nor has it been applied to a complex virtual environment. Barot et al. [2013] have described the use of activity models in virtual training environments, enhancing them with various models of error generation. By contrast, we are aiming at unifying task description, situation generation, and error recognition through a single knowledge representation.

Amir and Gal [2011] present a solid validation of task recognition in virtual laboratories with an interactive application that recognizes the user's current task. Task recognition is performed using grammar rules that represent tasks. They evaluated this method and proved it to be effective. The main difference to our approach is how we represent the possible hierarchical plans. We use an explicit task network decomposed *a priori* instead of an implicit representation based on grammar productions. One advantage of our explicit representation is that it can be represented visually, thus easing the procedure management. Another important difference is that our system provides real-time assistance to the users during the simulation.

Another application of these techniques is a framework built to assist in the development of systems for the recognition of high-level surgical tasks using analysis of videos that were captured in an operating room [Lalys et al. 2012]. Here, the processing is based on dynamic time warping and hidden Markov models. As processing occurred after all the data was collected, it is not a real-time application. Also, plan recognition was used in a domain in which users engage in exploratory and error-prone behaviors [Gal et al. 2012]. In this work, constraint satisfaction algorithms were used as a viable and practical approach for plan recognition in an educational software application similar to our intelligent biohazard training system.

2.3. Gesture Recognition

In the literature, there is a growing body of works dedicated to natural gesture–based interaction [Kristensson et al. 2012; Song et al. 2012]. Gutierrez et al. [2010] use a dynamic setup of haptic devices and motion capture for training in industrial maintenance and assembly tasks. The tasks that they intend to teach to users involve high precision and coordination, which justifies their need for a more complex setup than ours. We only need a Kinect sensor for detecting gestures, and high precision is not required in our system. For the same reason, we did not apply a complex technique as the one used by Oikonomidis et al. [2011], who investigated hand-articulation tracking in near real time using only visual information. This was achieved by formulating an optimization problem that minimizes the discrepancy between the 3D hand structure and the appearance of the hypothesized 3D hand model. While our gesture detection is not as precise, it is sufficient for our purpose and, most important, it is in real time.

In our case, we also did not have to adapt the recognition system to different users, as done by Hasanuzzaman et al. [2007]. The authors try to recognize that different gestures might have the same meaning in different cultures—for instance, the gesture recognized as "OK" is not the same in every country—but the gestures that we are recognizing are not attached to any culturally dependent meaning: everyone needs to close and open the hand when grabbing and dropping objects. The only gesture that could have a meaning is the one asking for help. We avoid this problem by giving specific instructions to the user on what gesture they should use.

Unzueta et al. [2008] propose a way to distinguish between static and dynamic gestures, that is, gestures that require one step and gestures that require more than one step, respectively. In our system, there is only one case of a dynamic gesture: moving the hand back and forth for using disinfectant. However, only one of our static gestures might be confused with this: the help gesture. Because this is a straightforward case, we opted for a much simpler alternative to the method described by Unzueta et al.

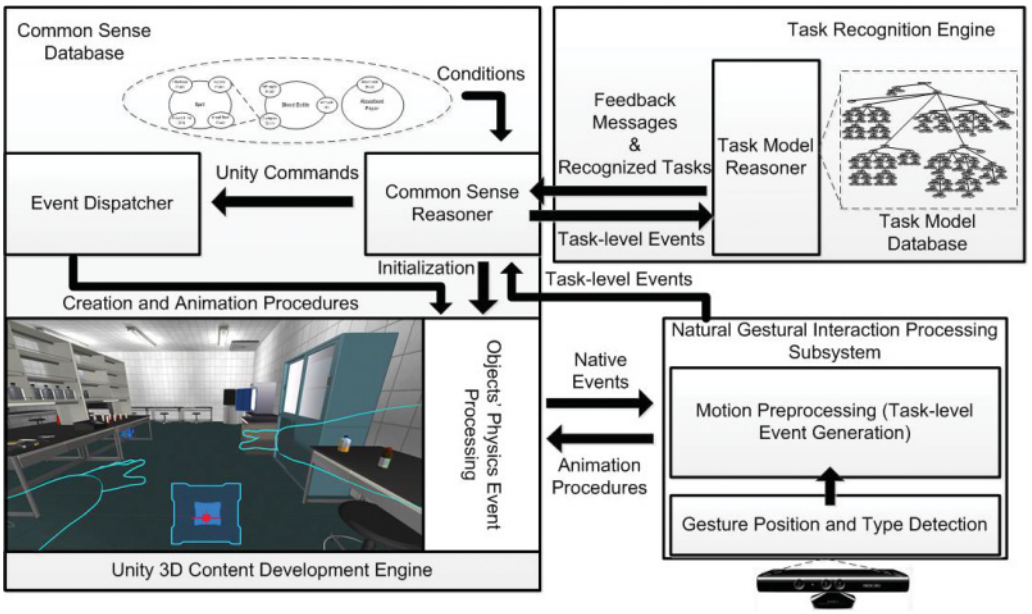

Fig. 1. System architecture. The knowledge-based module that performs task recognition and advice generation interfaces to the virtual world via an event-based system. These events are generated using the semantic properties of objects, and are triggered by physical interactions with these objects.

[2008]. We simply consider that the user is performing the help gesture if the user maintains it for more than 2s.

3. SYSTEM ARCHITECTURE AND IMPLEMENTATION

Our system recreates a biohazard laboratory (Level 2) from a user-centered perspective and supports realistic gesture-based interaction in a way that reflects real-world procedures. The use of a 3D content development engine (Unity3D[1]) provides a unified mechanism for visualization and interaction by taking advantage of the built-in mechanisms to define the physical behavior of objects. The knowledge-based aspects of the system consist of a representation of biohazard training procedures as task models, which can be instantiated in real time via the recognition of task-level events. In turn, task-level events can be derived from gestural interactions between the user hands and laboratory objects based on semantic properties attached to objects and common sense reasoning. The system architecture is shown in Figure 1.

3.1. The Interaction Processor

The interaction processor provides two types of input methods:

—Keyboard and mouse device input as a traditional method
—Gesture-based interaction using Kinect as an alternative method

In training scenarios in which physical interaction is important, gesture-based interaction has great potential as an object manipulation mechanism. It may support natural task execution as gestural interactions accurately mimic task-level events in the real world. Additionally, it may enhance the immersive attribute of the whole virtual

[1]http://unity3d.com/.

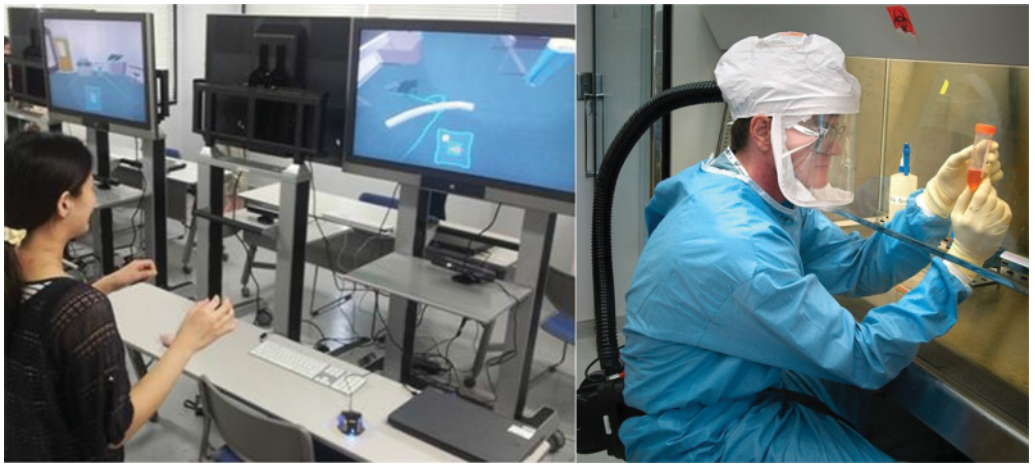

Fig. 2. Real-world lab work involves extensive and careful use of a person's hands. Likewise, our system emphasizes this fact by implementing a Natural Gestural Interaction Processor using Kinect. The virtual hands in the system represent the user's avatar controlled by Kinect. Lab technician picture rights provided by the Centers for Disease Control and Prevention's Public Health Image Library.

environment [Sridhar and Sowmya 2008; Lu et al. 2012]. Based on these considerations, we implemented a gesture-recognition framework using Kinect that allows users to interact naturally with the 3D environment. By using gestures, users would be able to accurately mimic real-world actions, particularly in terms of "taking," "carrying," and "releasing" virtual objects.

While gesture-based interaction is used in our setup, it is not the focus of our research. Thus, we had to compromise on some aspects of the interaction. First, the version of Kinect used was the XBox version of the device, which demonstrated some spatial constraints for users. For instance, the tracking of the user's body and hands cannot be closer than 1.6m. Second, if someone other than the user entered in Kinect's area of effect, which expands to 2.5m, Kinect started to detect both of them and malfunction until the other user left. Despite these issues, the device had a high degree of robustness when used in the appropriate (albeit slightly constrained) range of detection and environmental conditions.

3.1.1. Implemented Gestures and Corresponding User Actions. There are two main types of gestures that needed to be defined in order to cover all operations in the biohazard laboratory: (1) gestures to navigate inside the 3D environment, and (2) gestures to manipulate objects, corresponding to the *take object*, *drop object*, *use object*, *open object*, and *close object* user actions. Execution of these gestures is indicated by showing two virtual hands and arms that mimic the detected gesture (see Figure 2).

Additionally, if the system detects that the user's hand is over an object, it will highlight it and show a hand suggesting the grab gesture (see Figure 3(a) below). The goal of this feature is to better inform the user on the available actions at any given moment.

The design of navigation gestures was a challenge because we needed to define a natural way for users to indicate motion in the virtual world while remaining inside a limited area in the real world. Thus, for moving around the virtual world, we decided to implement a control scheme based on the placement of the user within two squared areas (see Figure 1). While the user is inside the inner area, the avatar does not move. If wishing to move in any direction, the user simply needs to step outside of the inner area in that same direction. The further the user moves in that direction, the faster

the avatar will move in the virtual world. The outer area is used both as a way to measure the velocity at which the avatar should move and as a way to inform the user of the limitations of one's movements. If the user steps out of the outer area, the Kinect motion detection stops being accurate. For that reason, we stop all motion detection waiting for the user to step in the recognition area again.

Likewise, the gestures to manipulate objects were implemented trying to maximize the naturalness of the actions themselves. We are detecting two main gestures, closing hand and opening hand, along with a third one, moving a hand quickly back and forth.

Based on these physical hand gestures, the following types of actions were implemented.

—A "take object" action, in which the user closes the palm over the object. The object will remain attached to the virtual hand as long as the user keeps the hand closed.
—A "drop object" action in which the user opens the palm while holding an object.
—A "use object" action in which the user moves one's hand back and forth for a while over an object, for instance, when the user needs to use a disinfectant object to clean a toxic spill.
—An "open object" action (same user hand gesture as for "take object"). Object (e.g., refrigerator) opens when closed.
—A "close object" action (same user hand gesture as for "take object"). Object closes when open.

We note that in our system, hand gestures are contextual, that is, the same physical gesture can refer to different actions in the 3D environment depending on the context (the type or state of an object). For instance, the user may take an object by closing one's hand while pointing at it; similarly, the user may use this gesture to open or close a door. The user may grab an object by maintaining a closed hand. The disinfectant (see list) can be used by grabbing it and moving it above the area that has to be disinfected.

Informal testing demonstrated that these methods are the most user-friendly for people operating inside the virtual environment through the exclusive use of their hands. As the majority of actions in the system can be carried out by closing, opening, or moving a hand, users require only a little amount of time and effort to get used to this interface.

3.1.2. Mouse and Keyboard Interface. We also implemented a mouse and keyboard interface to be able to compare it to the Kinect interface. We tried to reproduce as much of the Kinect interface as possible so that comparing both methods would be reliable. For this reason, all actions that the user can perform using Kinect can also be performed using mouse and keyboard:

—Pressing the arrow keys or "WASD" moves the avatar in the virtual world.
—Moving the mouse changes the position of the avatar's hand. In this version of the interface, the user controls only one hand as it would be impractical to control two hands with one mouse. As with the Kinect interface, if the hand reaches the limits of the screen, the view rotates in that direction.
—Left-clicking the mouse is the equivalent to the close-hand gesture. Therefore, it can have the effect of taking, opening, or closing an object, depending on the context.
—Right-clicking the mouse is the equivalent of the open-hand gesture. Therefore, it has the effect of dropping an object if one is currently held.
—Moving the mouse back and forth when holding a disinfectant has the effect of using the disinfectant.

3.1.3. Task-level Event Generation from User Actions. The data capture and gesture recognition processes are performed by the Gesture Position and Type Detection component

(see Figure 1). It was implemented as an extension of the official Microsoft Kinect framework. Upon the recognition of a gesture, the Motion Preprocessing component determines the appropriate task-level event based on heuristics regarding the placement and intention of the user when performing the detected gesture.

A task-level event is an action that is indivisible from a task perspective and used to perform task recognition and prediction processes on the task model. The task-level event is then passed to the Common Sense Reasoner to analyze its effect on the virtual environment. In the case in which a task-level event is not generated from a recognized gesture (e.g., navigation gestures), the gesture is sent directly to the Objects' Physics Event Processing component as an animation procedure.

Task-level events corresponding to the actions carried out by the user are formalized as combinations of gestures and objects in the virtual environment (that are targeted by the gestures). This approach supports a consistent definition of a set of user actions from primitive gestures and the classification of objects and their properties. In turn, the recognition of a task-level event serves as the activation of physical behavior corresponding to the events' consequences, or modifications to the virtual environment state. Recognition of task-level events is based on predefined heuristics regarding

—the type of gesture;
—the target object, that is, the object that will be manipulated;
—the surface object, that is, the object over which the target object is to be manipulated;
—the object already present in the gestured hand; and
—the duration of the gesture—SHORT_DURATION (1s), LONG_DURATION (2-3s).

To determine the target and surface objects, we implemented a simple ray-casting technique in which a target or surface object is selected if it is close enough to the user. More advanced techniques for 3D object manipulation have been implemented, for example, in Bowman and Hodges [1997]. However, we decided to use a simple method because (1) we wanted to emphasize the navigational aspect by letting users approach the objects themselves, and (2) in-hand manipulation of objects was not a necessary feature in our environment.

The user-generated task-level events currently implemented in our system are:

—*Take event*: It occurs when the user closes one's palm for a short time over an object that can be carried away ("take object" user action). If the gestured hand is empty, the event is generated as `Take(TARGET_OBJECT)`. If the gestured hand is carrying another object, the component determines if the carried object can be used to carry a second object. If that is the case, then the event is generated as `Take(TARGET_OBJECT, MEANS_OBJECT)`.
—*Open event*: It occurs when the user closes one's palm for a short time over an object that can be opened ("open object" user action). If this object is not opened yet, the event is generated as `Open(TARGET_OBJECT)`.
—*Close event*: Similar to the Open event, but the object needs to be opened this time ("close object" user action).
—*Use event*: It occurs when the user moves one's hand for a long time over an object (the surface object), having another object in the gestured hand ("use object" user action). If the latter can be used over the former, then the event is generated as `Use(TARGET_OBJECT, SURFACE_OBJECT)`.
—*Drop event*: It occurs when the user opens one's palm for a short time while having an object in the gestured hand ("drop object" user action). The event generated here is `Drop(TARGET_OBJECT, SURFACE_OBJECT)`, with SURFACE_OBJECT being the object over which the TARGET_OBJECT will be dropped.

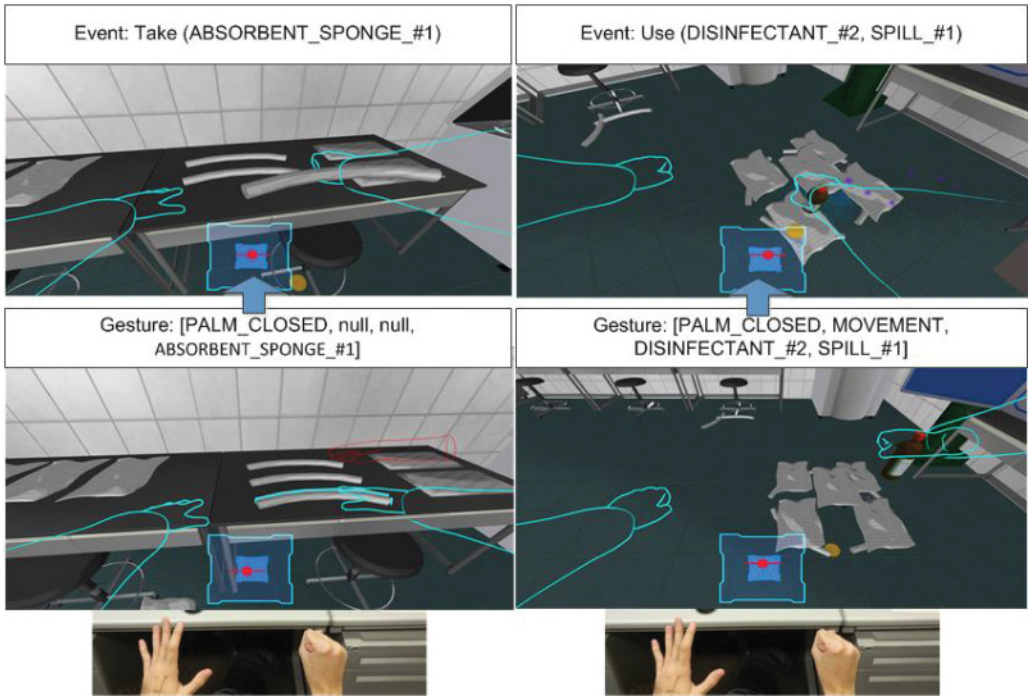

Fig. 3. The "Take Object" and "Use Object" Action-Event Generation Sequence: (a) when the system detects the PALM_CLOSED gesture over the ABSORBENT_SPONGE object, the `Take` event is generated; and (b) when the system detects the PALM_CLOSED gesture and is moving over the SPILL object using the DISINFECTANT object, the `Use` event is generated.

For instance, when a "take object" action has been detected, the Motion Preprocessing component receives the tuple ⟨PALM_CLOSED, SHORT_DURATION, OBJECT_ID, null⟩, and determines that a `Take` task-level event has taken place (see Figure 3(a)). In the case of a "use object" action, it receives the tuple ⟨PALM_CLOSED, LONG_DURATION, OBJECT_ID, SURFACE_ID⟩, and determines that a `Use` task-level event has taken place (see Figure 3(b)).

3.2. The Common Sense Reasoner

When the Natural Gestural Interaction Processor sends a task-level event to the Common Sense Reasoner, this module determines the physical consequences that the aforementioned event generates inside the virtual environment. These physical consequences are instantiated through the inference on semantic relational conditions defined between the objects in the environment (see Figure 4). For instance, if a user dropped a bottle on the floor, the Common Sense Reasoner would generate the physical consequences of such an event based on the physical characteristics of the bottle and the substance it contains. The rationale for using a Common Sense approach is that the actual detailed simulation of some events is not always relevant (e.g., spill progression or actual shattering of a vial). On the contrary, it is important to maintain an updated symbolic representation of the system (e.g., vials intact or broken, liquid inside or outside the vial).

More advanced models for common sense reasoning rely on low-level physics engine directives to perform the processing of physical consequences (e.g., Lugrin and Cavazza [2007]). In our case, the Common Sense Reasoner uses task-level events as directives.

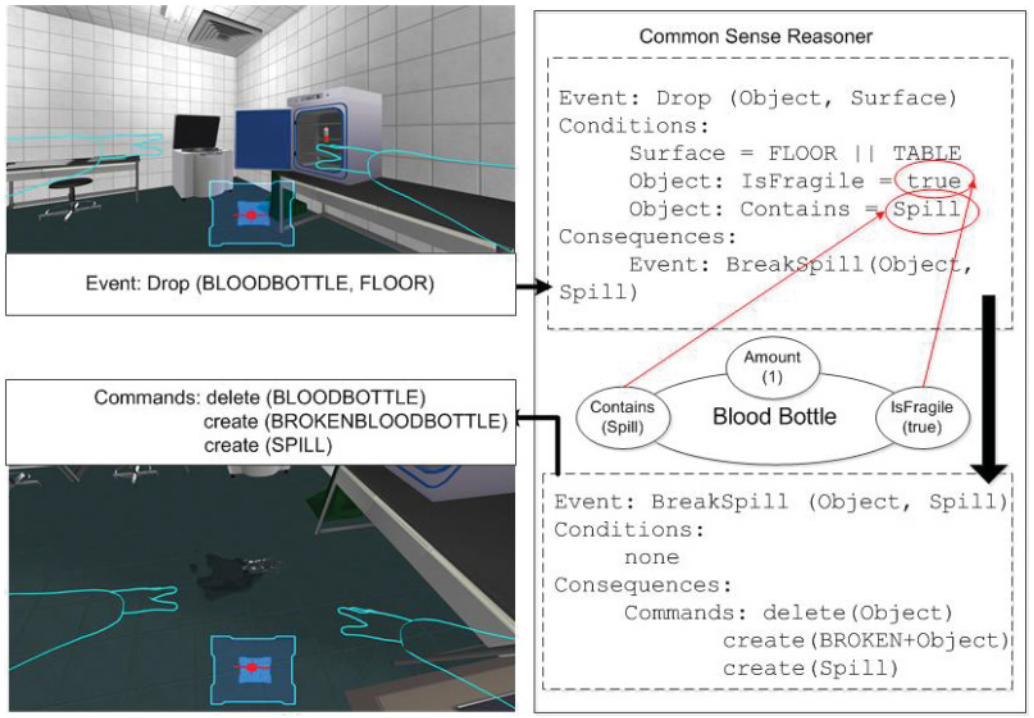

Fig. 4. The Common Sense Reasoner: Detailed process for the `Drop(BLOODBOTTLE, FLOOR)` event. The Common Sense Reasoner bypasses low-level physical simulation while updating the physical state of objects in the knowledge base. It interfaces to the Unity3D engine to update the physical appearance of objects following a change in state.

Since our system is focused on the achievement of realism from the perspective of training scenarios, the implementation of a specialized physics engine was not necessary. However, our Common Sense Reasoner essentially implements a Qualitative Physics approach [Cavazza et al. 2004]. This scheme supports better reasoning on the causal aspects and maintains an explicit representation of the action's consequences. As physical accuracy is of less importance in our application, the integration with Unity's native physics engine was not necessary.

Task-level events are processed through a series of cascading rules, which are defined in the Common Sense Database (see Figure 1). Objects specified as parameters of those rules are matched to the parameters of the task-level events and the semantic properties of those objects, which are also defined in the database. Once a task-level event is completely processed, the Common Sense Reasoner interfaces to the low-level primitives of the Unity3D engine, which supports the interactive visualization of the world state: in particular, the creation or deletion of objects, or changes in their visual appearance. These commands are received by the Event Dispatcher component, which executes the commands.

3.3. The Task Recognition Engine

The Task Recognition Engine performs task recognition and prediction procedures based on the task-level events that were received from the Common Sense Reasoner. These procedures are explained in the following section.

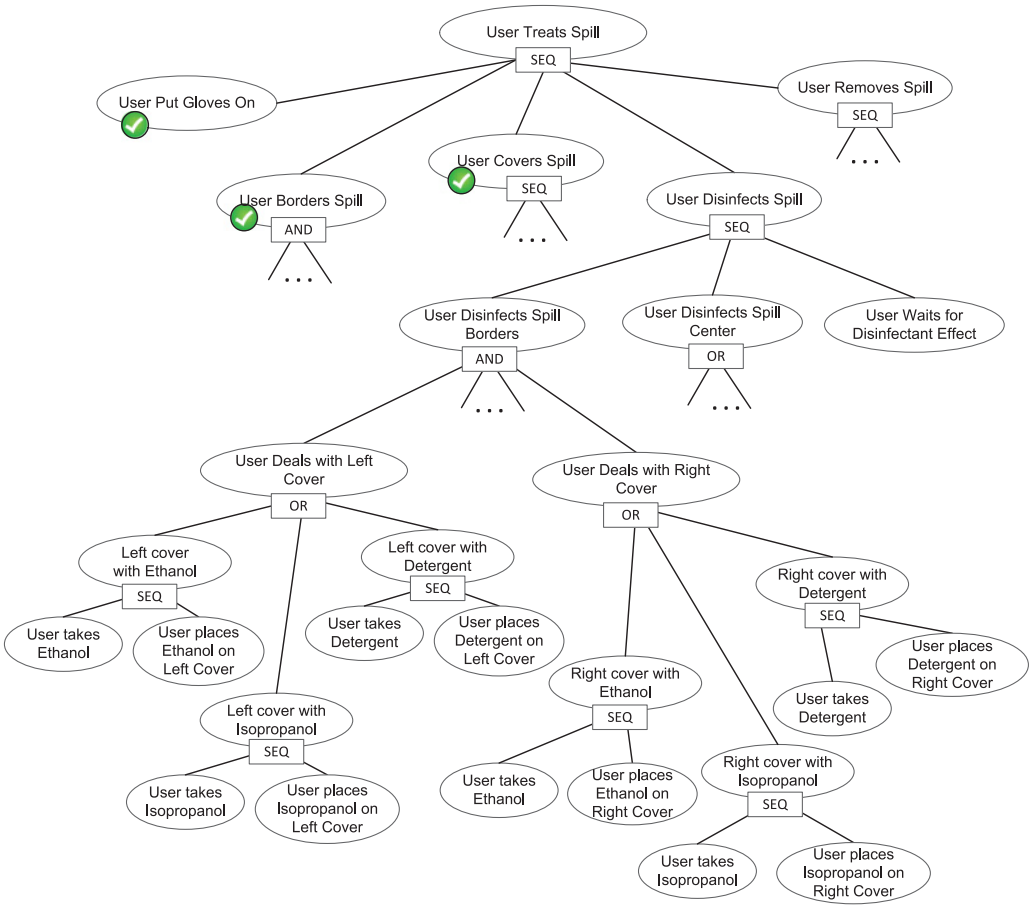

Fig. 5. Part of a task tree modeling the training procedure that has to be followed for the treatment of a toxic spill. The complete task model contains more than 250 nodes and requires at least task-level user actions to be completed.

4. TASK-BASED REPRESENTATION OF BIOHAZARD PROCEDURES

The main purpose of the Task Recognition Engine is to provide task-related assistance to users during the simulation. In our context, students have been studying security procedures by reading a textbook and are now prepared to rehearse them in the virtual laboratory. Thus, the purpose of the system is to provide assistance to students when necessary by monitoring their actions. For that reason, we opted for a simple knowledge-based representation of biohazard training procedures that supports real-time monitoring from user actions and provides real-time assistance and guidance in case of incorrect execution. Furthermore, our approach is compatible with postprocedure debriefing, by tracing the various errors and aborting the simulation when the situation cannot be corrected.

Biohazard training procedures are represented using a hierarchical task model, as shown in Figure 5. Tasks on top of the tree represent *abstract tasks* that are decomposed into simpler subtasks until reaching the leaves of the tree, which correspond to *task-level events* or basic actions that the user performs in the virtual environment. In this way, training procedures are represented as a *task tree* in which the internal-node group

subtasks in three different ways: *AND* (all subtasks must be completed in any order), *OR* (different ways to complete a task) and *SEQ* or sequences (all subtasks must be completed in the proper order). Note that some of these procedures are quite complex: the complete task tree that describes the protocol to treat the spill of a hazardous substance contains more than 250 nodes and requires at least 40 task-level user actions to be completed.

This knowledge representation is inspired from a hierarchical planning formalism known as *Hierarchical Task Networks* (HTNs) [Erol et al. 1994]. Actually, the task tree is similar to an explicit HTN in which the main task has been decomposed *a priori* and entirely, down to the level of elementary actions, rather than being dynamically refined using decomposition methods [Nau et al. 2004]. Thus, instead of representing security protocols as a collection of refinement methods, we use an explicit task tree that can be represented visually, facilitating knowledge elicitation. Explicit representations tend to be common when representing protocols when a visibility over the protocol is required both at knowledge elicitation time and during instantiation at training time [Georg and Cavazza 2007]. By contrast, refinement methods tend to be used when abstraction between hierarchical levels is more relevant. Explicit representations also facilitate the inclusion of ordering constraints on actions and explicit representations of common errors that are attached to specific situations.

Note that we do not use this task tree for plan generation and, in fact, we do not use any planning technique. We use a hierarchical task model (1) to intuitively represent training procedures as multistep decomposable processes, and (2) to perform task recognition, which is achieved by traversing the tree each time the user performs an action in the virtual environment. In general, the idea of using planning-based formalisms to represent procedural knowledge is not unusual even in systems that do not use planning algorithms [Bradbrook et al. 2005; Shahar et al. 1998].

There are several benefits in using an approach inspired by HTNs to model complex procedures. First, HTNs have been shown to be an effective way to encode domain knowledge and to restrict the order in which actions can be combined [Nau et al. 1998; Wilkins and desJardins 2001]. Second, HTNs are intuitive enough for experts and allow them to work at different levels of abstraction, which decreases the effort required to model complex activities [Currie and Tate 1991; Muñoz-Avila et al. 2001]. Along the same lines, HTNs promote reusability of abstract tasks among different protocols since subtrees can be shared in different branches of the tree. Finally, as we will describe in more detail later, the hierarchical structure can be exploited to detect incorrect actions and provide real-time feedback to the user. The task trees were created during extended sessions with the domain experts, specifically Dr. Mika Shigematsu, our coauthor from NIID.

4.1. Recognizing Correct Actions

To monitor user activity, we need to identify the actions that the user performs as the student progresses through the biohazard training protocol. In this section, we describe how to identify actions that are correct according to the task model; in the following section, we will explain how to detect and help the user when the student makes a mistake.

It is important to note that the users of our system are students that have previously studied the laboratory procedures in textbooks. Thus, we expect only small deviations from the correct procedure during the simulation but not random meaningless actions or users with goals very different from the ones involved in the procedure. In this sense, we do not require a plan-recognition algorithm to tell us the most likely goal the user is currently pursuing, but rather some way to monitor whether the user is following the procedure correctly or if the user has performed some mistake or skipped some important step.

In order to provide feedback to the user during the simulation, our system needs to recognize the user actions in real time. We assume some simplifications in the task tree representation:

—We consider variables from different leaf nodes to be independent. That is, we cannot use a common variable to represent that the object that was used in one action is the same one that will be used in another posterior action. Sibling trees in the structure are independent.
—Task-level events representing the user actions are forwarded to *all* the subtrees in AND and OR nodes, that is, one user action can instantiate different leaf nodes.
—We use a greedy approach in which, once a tree node has been instantiated, its status no longer changes.

These simplifications allow us to avoid the nondeterministic choices usually involved in plan-recognition problems; thus, we can instantiate the task tree very efficiently. However, they also limit the type of plans that we are able to recognize or differentiate. For example, we cannot accurately recognize plans in which the same action is executed several times consecutively. Usually, these limitations can be overcome by carefully designing the task-level events and the task tree. In the worst case, the task tree will be an approximation of the laboratory procedure and the system will not be able to detect all the user errors. Systems requiring a more precise representation can use other types of plan-recognition algorithms based on hierarchical task trees [Geib and Goldman 2009; Kabanza et al. 2013] at the cost of the computational complexity of those algorithms.

Algorithm 1 describes our approach. The Task Model Reasoner receives task-level events representing the actions that the user has performed in the virtual environment and propagates those events tothe leaves of the task tree. When one of those events matches the action contained in one leaf node of the tree, the reasoner instantiates the corresponding task and, probably, other higher-level tasks depending on it. In this way, the task tree is instantiated from the leaves to the root as the user advances in the simulation.

There are four different types of nodes in the tree: AND, OR, SEQ, and LEAF. When an inner node receives the task event, it propagates the event to its direct children, then checks if it has to instantiate itself according to its type. Note that AND and OR nodes propagate the event to *all* its children and that the order in which they are traversed is not important to instantiate the current node because we consider each subtree independently from its siblings. SEQ nodes, on the contrary, propagate the event to their children in a specific order until one of them is not instantiated.

The matching between task-level events and leaf nodes of the tree is quite straightforward. Task-level events describe specific actions performed in the virtual environment; thus, they cannot contain variables. For example, the task-level event `Take(ethanol_#1)` is received when the user takes a bottle of ethanol represented with the symbolic constant `ethanol_#1`. Leaf nodes of the task tree, on the other hand, describe actions using typed variables. For example, the leaf node *User takes ethanol* contains the action `Take(?x1 - Ethanol)` that accepts any object of type `Ethanol` and therefore matches the previous user action.

This simple bottom-up instantiation of the task model from user actions supports the analysis of the user behavior from the perspective of the task to be learned, supports a unified mechanism to assess the user, and provides real-time feedback and assistance during the execution of the most complex procedures.

For example, the tree in Figure 5 shows part of the training protocol that must be followed for the treatment of a toxic spill. In this case, the original task *User Treats Spill* is decomposed into five subtasks that must be performed in order: (i) Put on the

ALGORITHM 1: Algorithm to Instantiate Nodes in the Procedure Task Tree from the Events Produced as a Consequence of the Student's Actions.

```
def ProcessEvent1(inout treeNode, in event):
    /* treeNode: Semi-instantiated task tree                                  */
    /* event: task-level event representing some user action                  */

    /* If the node is already instantiated there is nothing to do             */
    if treeNode.instantiated return

    if treeNode is AND node
        /* Pass event to subtrees                                             */
        for stn in treeNode.subtrees()
            ProcessEvent1(stn, event)
        treeNode.instantiated ← AllInstantiated(treeNode.subtrees())
    elif treeNode is OR node
        /* Pass event to subtrees                                             */
        for stn in treeNode.subtrees()
            ProcessEvent1(stn, event)
        treeNode.instantiated ← AnyInstantiated(treeNode.subtrees())
    elif treeNode is SEQ node
        /* Pass event to subtrees in order. If some subtree is not instantiated then
           leave                                                              */
        for stn in treeNode.subtrees()
            ProcessEvent1(stn, event)
            if not stn.instantiated return
        treeNode.instantiated ← True
    elif treeNode is LEAF node
        /* Matching between the event and the node action                     */
        tree.instantiated ← Match(treeNode.action, event)
```

Gloves, (ii) Border the Spill, (iii) Cover the Spill, (iv) Disinfect the Spill, and (v) Remove Spill. Figure 5 shows the current state in which the user has successfully completed the first three subtasks, and now, the student has to disinfect the spill.

On the other hand, Figure 6 shows an example of the interaction with the system and the task instantiation process that the reasoner performs. At this stage of the procedure, the user has to disinfect the spill that was previously covered and choose among different chemicals depending on the nature of the spill. In Step 1, the user chooses to take a bottle of ethanol (represented as the formal object `ethanol_#1` of type `Ethanol`), an appropriate disinfectant, so that the task reasoner receives a `Take(ethanol_#1)` event. The reasoner traverses the current task tree looking for leaf nodes accessible in the current state that contains a compatible task. In this case, the reasoner finds several leaf nodes with the task `Take(?x1 - Ethanol)` that matches the user action. The reasoner instantiates the tasks and sends back an event indicating that the action was recognized.

Next, in Step 2, the user utilizes the same bottle with ethanol to disinfect the left side of the spill. In this case, three different task nodes are instantiated as a result of the action (see Figures 5 and 6), because this action completes two other higher tasks in the task tree. Note that one action might trigger the recognition of several tasks at different levels in the tree.

4.2. Providing Assistance

Deciding *when* and *how* to help the user is a very complex problem that is beyond the scope of this paper. However, the logical structure of the task representation provides a

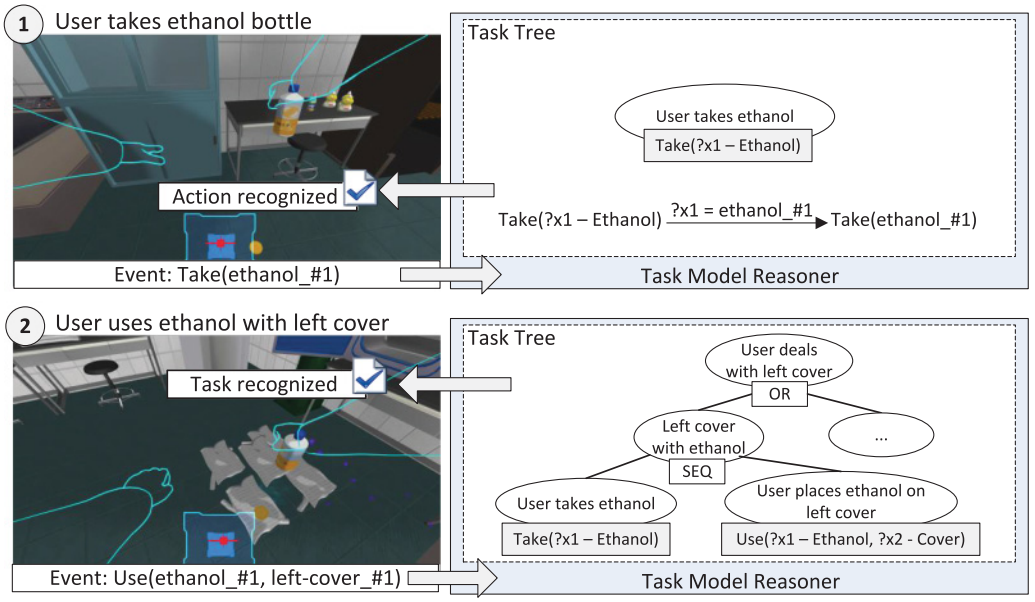

Fig. 6. Example of task instantiation as the user progresses in the procedure (Part 1). In Step 1, the user performs an action that is recognized as correct because it matches the task contained in a leaf node. In Step 2, the user performs another correct action that triggers the instantiation of several tasks at different levels in the tree.

basic mechanism to reason on the user progression that can be used to detect incorrect actions and provide useful hints. In this sense, our system can be called a training or rehearsal system.

Basically, we can detect two different types of incorrect actions:

—Actions that are explicitly represented in the task model as incorrect choices, and
—Actions that are part of the correct procedure but should not be executed yet.

The first type corresponds to those errors that have been anticipated by the experts: for example, to use a wrong object like dropping an absorbent pad over the spill, or to use the wrong instance of the correct object like using a disinfectant that is not suitable for the kind of spill being treated. These errors are instances of task events received from the Common Sense Reasoner and are represented explicitly in the task tree as error nodes. These error nodes can contain specific messages to explain why the user action is not adequate. For example, if the user tries to use an inadequate disinfectant, the system can provide a warning and explain why the student should not use that chemical in this situation.

Algorithm 2 shows a more elaborated version of the task instantiation algorithm from previous section, to take into account the existence of these error nodes. The *errorTask* parameter in the algorithm is used to mark when one of these error nodes is instantiated as a result of some user error. When one of these errors is detected, the process of instantiation is interrupted, the Task Recognition module informs of the error and then removes the error node from the instantiated tree. The *newTasks* parameter returns the new correct tasks that have been instantiated as a result of processing the task-level event.

The second type of mistake comprises those actions that should not be executed at present, because they are part of a *SEQ* node with some previous subtask that is

ALGORITHM 2: Algorithm to Instantiate Nodes in the Procedure Task Tree from the Events Produced as a Consequence of the Student's Actions

```
def ProcessEvent2(inout treeNode, in event, out errorTask, inout newTasks):
    /* treeNode: Semi-instantiated task tree                                  */
    /* event: task-level event representing some user action                  */
    /* errorTask: new instantiated error task node                            */
    /* newTasks: new instantiated correct task nodes                          */

    /* If the node is already instantiated there is nothing to do             */
    if treeNode.instantiated return

    if treeNode is AND node
        /* Pass event to subtrees. If some error node is instantiated then leave */
        for stn in treeNode.subtrees()
            ProcessEvent2(stn, event, errorTask, newTasks)
            if errorTask != null return
        treeNode.instantiated ← AllInstantiated(treeNode.subtrees())
    elif treeNode is OR node
        /* Pass event to subtrees. If some error node is instantiated then leave */
        for stn in treeNode.subtrees()
            ProcessEvent2(stn, event, errorTask, newTasks)
            if errorTask != null return
        treeNode.instantiated ← AnyInstantiated(treeNode.subtrees())
    elif treeNode is SEQ node
        /* Pass event to subtrees in order. If some error node is instantiated or
           some correct node is not instantiated then leave                   */
        for stn in treeNode.subtrees()
            ProcessEvent2(stn, event, errorTask, newTasks)
            if errorTask ≠ null or not stn.instantiated return
        treeNode.instantiated ← True
    elif treeNode is LEAF node
        /* Matching between the event and the node action                     */
        tree.instantiated ← Match(treeNode.action, event)

    if treeNode.instantiated
        if treeNode.isError
            errorTask ← treeNode
        else
            errorTask ← null
            newTasks ← newTasks ∪ {treeNode}
```

not achieved yet. For example, according to the Spill Disinfection procedure, the user should not try to disinfect the center of the spill until the borders have already been disinfected.

Algorithm 3 explains how to detect those errors. Note that this algorithm is used only when the task-level event did not instantiate any task node in Algorithm 2; thus, we know that the user action is not correct and it does not correspond to any error anticipated by the experts. First, we look for semi-completed SEQ nodes in the tree, that is, not instantiated sequential nodes with some descendants instantiated. They represent sequential steps in the procedure that have been started but are not finished. Those nodes are retrieved using a depth-first search; thus, they are sorted by depth. Then, we determine if some of its descendants are instantiated as a result of processing the task-level event; but, in this case, we check all the subtrees of the SEQ nodes. If

ALGORITHM 3: Finding High-Level Tasks that the User is Incorrectly Trying to Complete Because Some Previous Step in the Protocol has not been Completed Yet

```
def orderError(in treeNode, in event, out incorrectTask):
    /* treeNode: Semi-instantiated task tree                                  */
    /* event: task-level event representing some user action                  */
    /* incorrectTask: incorrect task that the user is trying to complete      */

    /* Semicomplete SEQ nodes in the tree sorted by depth (the deepest first) */
    SEQnodes = semicompleteSEQnodes(treeNode)

    for node in SEQnodes
        for st in node.subtrees()
            newTasks ← ∅
            processEvent2(st, event, errorTask, newTasks)
            if errorTask = null and newTasks ≠ ∅
                incorrectTask ← st
                return
    incorrectTask ← null

def semicompleteSEQnodes(in treeNode):
    if treeNode.isError or treeNode.instantiated or treeNode.type == LEAF
        return []

    SEQnodes ← []
    for stn in treeNode.subtrees()
        SEQNodes = append(SEQnodes, semicompleteSEQnodes(stn))

    /* We say that a SEQ node is semicomplete if it is not instantiated but has some
       descendants instantiated                                               */
    if semicompleteSEQ(treeNode)
        SEQNodes = append(SEQnodes, treeNode)
    return SEQnodes
```

some descendant node is instantiated, then we assume that the user is probably trying to do something that should not be done yet. Once we know which high-level task the user is trying to complete, we can use that information to interact with the user without having to make references to specific actions. Note that we iterate over the SEQ nodes sorted by depth so that the interaction with the user will be based on the more specific SEQ node in which we detect the problem.

Let us take into consideration Step 3 in Figure 7, in which the user decides to use ethanol on the center of the spill immediately after disinfecting the left border. In this case, the reasoner traverses the task tree, detecting that the action does not match any leaf node compatible with the current state, but rather a leaf node in a branch that should not be executed yet. In this way, the system may notify the user that the first thing to complete is *Disinfect Spill Borders*, which corresponds to the left not-instantiated sibling of the *Disinfect Spill Center* task. Note how the system takes advantage of the hierarchical task model to describe what parts of the training procedure the user should complete first without describing the specific actions to take.

Apart from the two types of errors that we have explained (anticipated by experts and skipped steps), the users can make other mistakes that we are not able to detect. In order to detect them, we would require a richer semantic description of the domain. However, it is relatively easy to add new error nodes to the task tree so that we can incorporate specific messages to deal with those new common mistakes when the experts detect them.

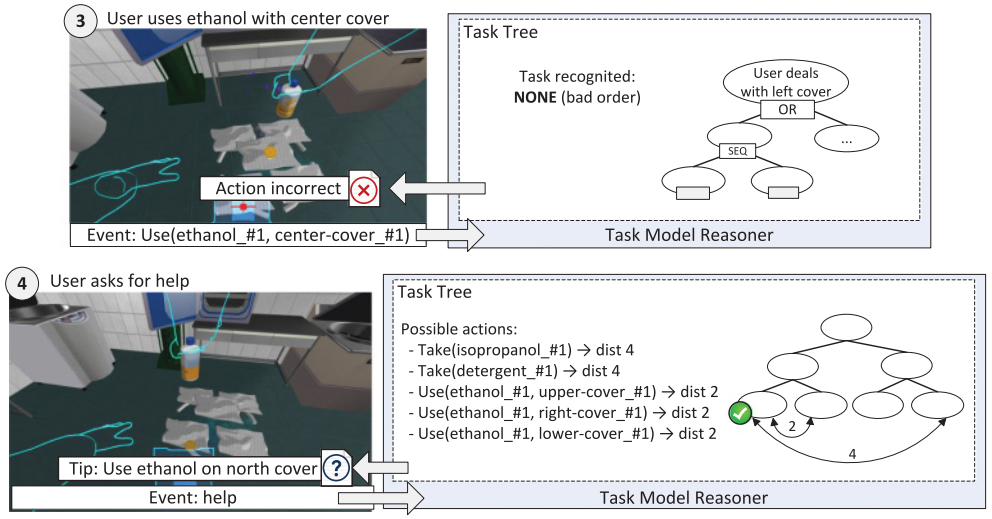

Fig. 7. Example of task instantiation as the user progresses in the procedure (Part 2). In Step 3, the user performs an (incorrect) action that should not be executed yet according to the procedure. In Step 4, the user asks for help; then, the reasoner computes the possible next actions, and selects which one to show using a heuristic approach based on distances in the task tree.

Finally, it may happen that the user does not know how to proceed next to deal with the remaining three border covers. In Figure 7, Step 4 shows what happens when the user asks for help. In this case, the task reasoner traverses the task tree looking for all the actions that can be executed next. Usually, there will be several different possible actions because the *AND* and *OR* nodes allow different execution paths.

In our example, users could either use the bottle with ethanol that they are holding in their hand with any of the three remaining borders, or they could take another disinfectant. Although all these actions are correct, some are more intuitive than others. In this case, the user is holding a bottle with ethanol that has already been used to disinfect a border; thus, we can presume that the user will want to use it again instead of taking another disinfectant.

In our system, we take advantage of the hierarchical structure of the task model to prioritize those possible actions in the tree that are closer to a task that has already been completed. We measure the distance of two nodes in the tree as the minimum number of edges between them. The intuition beneath this heuristic is dual: (1) the task hierarchy groups tasks that are part of a whole, that is, the task/subtask relation is a strong semantic constraint; and (2) if the user has performed a subtask, the user will probably try to complete the sibling subtasks in order to complete the higher-level task.

In the example, when the system computes the distances of the five leaf nodes that contain the possible actions closer to the completed task (*User takes ethanol*), it decides to prioritize the *User places ethanol on upper/right/lower cover* tasks because they are siblings (distance 2 in the tree). Thus, the system will advise the user to perform one of them.

4.3. Common Mistakes and Real-Time Feedback

The training system collects information about the errors committed by each of the students during the simulation. By analyzing this information, we can determine both specific weaknesses of each student and common mistakes made by most of them. The system also stores the times used to complete each task, which makes it easy to detect

the most complex tasks or those that most students have not assimilated properly. Section 5 reports on the details.

As explained earlier, the hierarchical task model that we use to represent the biohazard training procedures allows us to detect two types of errors in real time: (1) errors that are labeled as incorrect (by the domain expert) and (2) errors that result from the execution of actions at the wrong time. In both cases, the system is able to instruct the student by showing descriptive help messages, but using different approaches.

—*First case:* Both errors and their associated help messages are part of the domain model; therefore, they have been anticipated by experts.
—*Second case:* The system displays a generic message compelling the user to complete another task before executing the current action.

It is important to remark that, although the help message is generic, the task to complete usually corresponds to an internal node of the tree, that is, an abstract goal. In this way, the system is able to describe abstract goals that the user must achieve without indicating the specific actions to perform.

Regarding the first type of errors, the task model is able to detect when the user is placed in the wrong position, for example, between the spill and the air vent; when a wrong product is used to delimit or cover the spill, for example, absorbent pads; when the user chooses a disinfectant not suitable for the type of substance being treated; or when a product is handled with the wrong tool, for example, taking the waste with the gloves instead of the tweezers; among other incorrect actions. All of these errors are nodes in the task tree that are identified with a triggerable task-level event and each one is associated with a specific help message, as shown in Figure 12.

Regarding the second type of error, when the student performs an action that should not be executed yet, the system can detect several mistakes (an example is shown in Figure 7). In the following, we describe some of them and, in parentheses, the type of assistance message that the system produces:

—The student attempts to interact with some object in the laboratory, for example, to take the absorbent paper, before putting the gloves on (You should *Put the Gloves On* before attempting to *Border the Spill*).
—The student attempts to cover the center of the spill before covering all the borders (You should *Cover the Spill Borders* before attempting to *Cover the Spill Center*).
—The student covers the left border, then attempts to disinfect it without covering the rest of the spill first (You should *Cover the Spill* before attempting to *Disinfect the Spill*).
—The student attempts to dispose of the covers and then the borders in two different steps (You should *Merge the Waste* before attempting to *Dispose The Waste*).

In summary, the hierarchical-task model chosen to represent bio-safety lab protocols provides the following benefits:

—It is an intuitive and explicit representation that can be used and revised by domain experts.
—It enables real-time detection of two types of mistakes: (1) mistakes that have been anticipated by the experts and included explicitly in the model and (2) mistakes that arise when the student forgets some step of the protocol.
—It supports interaction with the student using abstract concepts represented by the internal nodes of the network. Thus, the system can inform the student about the next to-be-completed high-level task without referring to the contained basic actions.
—It can anticipate the most likely next action that the student will perform using a heuristic based on the hierarchical structure of the domain.

To the best of our knowledge, the combined use of these ideas in such interactive virtual training systems has not yet been investigated.

4.4. Complexity of Solution Space

In order to understand the intrinsic complexity of the procedure to manage a toxic spill used in the previous examples as well as the number of trainee errors that our model can detect, we performed additional experiments in which we automatically generated random sequences of user actions. In particular, because our objective was to analyze the complexity of search, we focused on the number of actions that our model is able to identify as correct or incorrect and the number of internal nodes in the task tree that must be traversed in order to identify each action.

These experiments are not meant to replace a staged user evaluation of the system as a whole: such an evaluation will be described in the next section. Instead, they contribute important information on system complexity, which not only provides a justification for the use of AI techniques but also gives indications on the scalability of the approach, both aspects differentiating it from scripted methods.

First, the graph in Figure 8 depicts the solution space of the simulation and indicates how nodes are instantiated.

Users can perform a full set of relevant actions in the virtual environment: move, take, drop, use, open, close (see Section 3.1). These types of actions are similar to planning operators that can be parameterized with different domain entities, resulting in about 2000 different ground actions when considering potential objects in the environment. From all of these, only around 25, can be really executed at a given time (i.e., their preconditions are satisfied in the current simulation state). If we take into account that in order to complete the toxic spill procedure, at least 46 ground actions are needed, it is easy to realize that the number of executable plans is just too large to try to find solution plans by generating random actions. Instead, we use the following approach: (1) we compute which ground actions are executable in each simulation state and how many of them are recognized as correct/incorrect by our task-recognition engine; (2) we select one of the correct actions and append it to the solution plan; (3) we execute the selected action, thus the simulation state changes, and repeat the process again until we find a solution plan. In other words, in this experiment, we use a greedy approach that does not consider nonmeaningful actions (actions that do not impact the goal task in any way) in the solution plans.

Figure 9 shows the number of executable actions in each simulation step. The chart shows average numbers obtained from 10000 different solution plans. The x-axis represents the current simulation step (an optimal solution plan consists of 46 actions), and the height of each bar indicates the number of executable actions in that step. We use 3 colors to represent how many of those executable actions are recognized as correct (blue), incorrect (red), and nonmeaningful (green). Note that although we compute all the executable actions in each step, only one of the correct actions will be executed in the virtual environment to reach the next simulation step. As can be observed in the figure, the number of correct actions always amounts to a small percentage of the executable actions. Regarding incorrect actions, the number of mistakes that we are able to recognize as such depends largely on the number of nodes explicitly asserted as incorrect in the task tree and the number of internal SEQ nodes. As a result, the task recognizer is able to detect several mistakes in some simulation steps and very few or none in other simulation steps.

Another interesting question for a real-time recognition engine is to measure the number of nodes in the task tree that need to be visited in order to detect a correct user action and instantiate a new tree task. Figure 10 shows that the percentage of visited nodes in each simulation step represents a small part of the tree (usually between

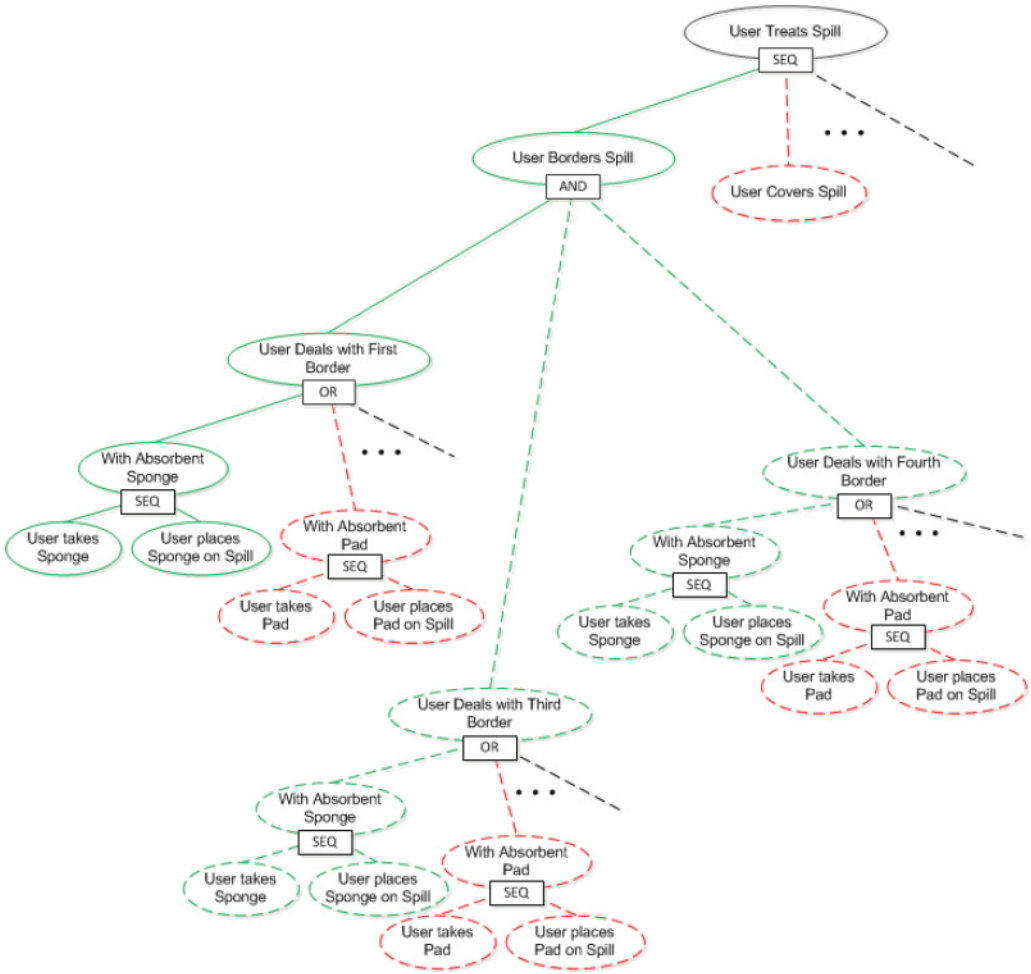

Fig. 8. Solution space of the simulation. Green nodes and lines represent correct user actions and red lines represent user mistakes. Black lines show solutions that cannot currently be explored due to constraints of the top-down search. Dashed lines represent nodes that are not (yet) instantiated, thus are future candidate actions for the user.

5%–20% of the total). This number depends basically on the types of internal nodes and the number of tasks already recognized. For example, an OR node implies that we need to search for the user action in each subtree while a SEQ node constraints the search to the next subtree. We also observe that the number of visited nodes tends to decrease as the solution plan evolves. Hence the number of nodes to be visited depends both on the structure of the task and the progression stage of the plan.

In our final experiment, we generated 1000 executable plans, each one with 100 random actions, and we counted the number of nodes in the task tree that were marked as completed after executing each plan. Figure 11 shows the percentage of plans that were able to complete at least n tasks (the number of completed tasks is in the x-axis). All the plans successfully completed the first task (User puts the gloves on), but less than 20% of the plans were able to complete 14 nodes in the task tree. There are 255

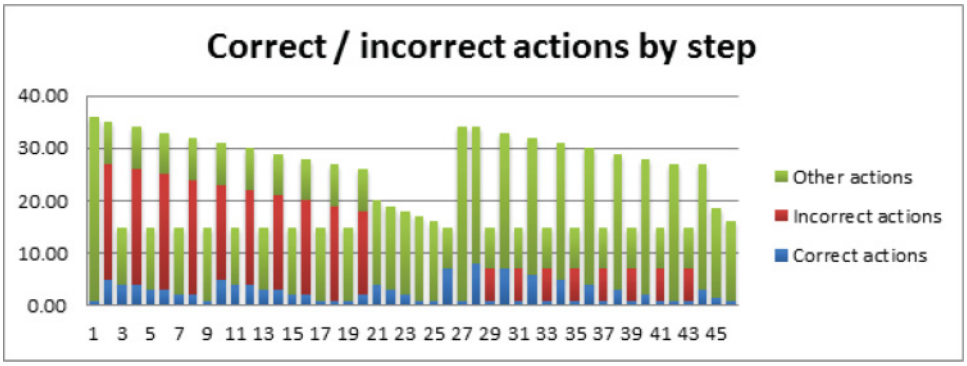

Fig. 9. Average number of correct/incorrect actions per step in each solution plan.

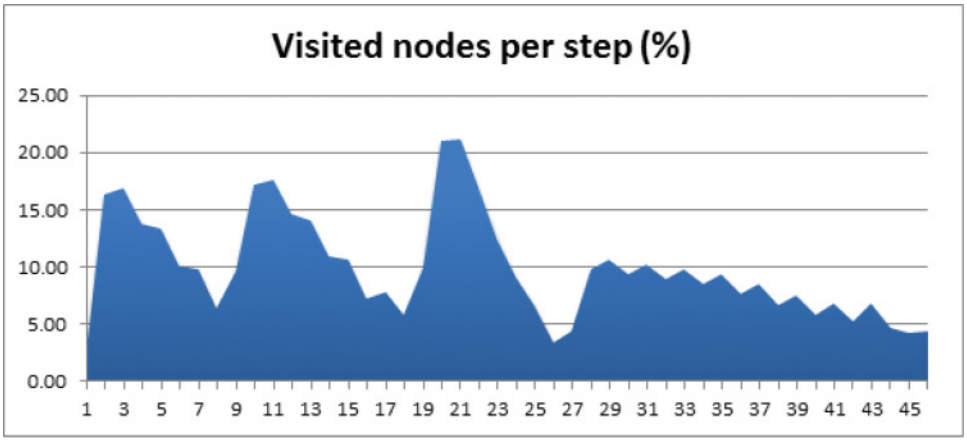

Fig. 10. Average number of nodes visited per step in a solution plan.

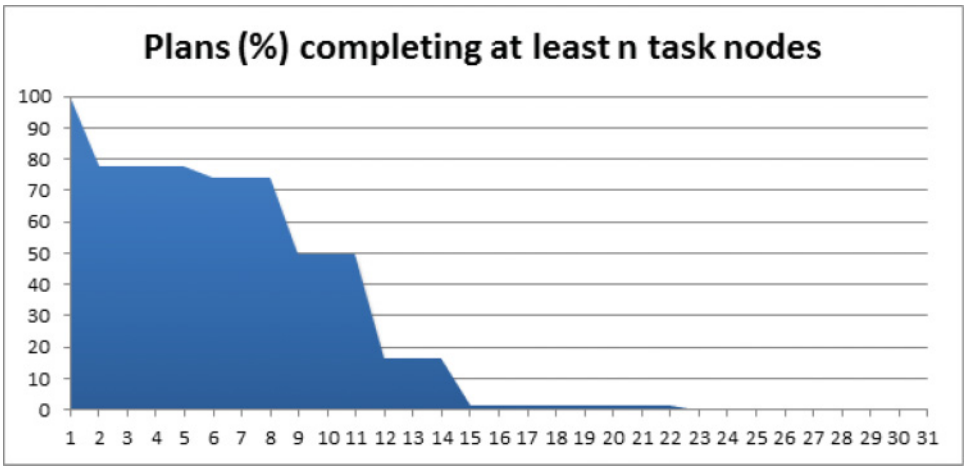

Fig. 11. Percentage of the plans completing at least n nodes.

nodes in the original task tree; thus, none of the random plans reached a significant stage. However, that is reasonable given the complexity of the domain.

By analyzing these results, we can confirm that the toxic spill protocol used in our experiments describes a very complex scenario in which the user can perform many different actions at every moment, but only a few lead to a successful solution. Therefore, the probability of completing the protocol by performing random actions is very low. We also confirm that the task-tree model presented in this article is able to represent complex protocols intuitively, and meets the requirements of a real-time task recognition system since only a small part of the tree needs to be visited to determine whether the user actions are correct or not. Finally, the results of our experiments in Kyushu University with real users, presented in the following section, will confirm that the task recognition system effectively helps the users to complete the protocol, tutoring them and correcting their errors throughout the simulation.

5. FIELD STUDY

We have conducted a field study with students from the medical campus of Kyushu University to answer three questions. *First*, we wanted to investigate whether real-time (dynamic) feedback allows for faster recovery from mistakes than (static) feedback. In the latter case, users open a screen that presents the protocol in text form. When opened, users cannot perform other actions until they close this screen. We hypothesize that dynamic feedback is more effective than static feedback.[2] *Second*, we wanted to assess whether a Kinect device or mouse device is more intuitive and easy to use by this user group. *Third*, we wanted to know whether dynamic feedback or static feedback is better regarding the recall of bio-safety lab protocols.

5.1. Method

5.1.1. Subjects and Design. The study had twenty-eight subjects, who were senior students of the medical laboratory technologist course, and a few freshmen from the graduate course. They were recruited since all had completed the clinical microbiology lecture and practice. We assumed that they understand the basic facility of a laboratory and "good microbial practice" in general. None had experience using 3D virtual-world technology or motion capture applications such as Kinect. In the study, we had twenty females and eight males between 19 and 22 (age average of 21.6). Subjects were paid an equivalent of USD 10 for participation. We prepared four conditions:

(1) "Kinect" condition: subjects use the Kinect device to perform predefined tasks in the 3D environment, such as grabbing an object and bringing it to another location, opening the door of a container, and so on. Those simple tasks are aimed at practicing the operation of the interaction device and are not related to resolving an accident in the bio-safety lab.
(2) "Mouse" condition: Subjects use the mouse device (and keyboard) to perform predefined tasks in the 3D environment.
(3) "Dynamic Feedback" condition: Subjects use Kinect and receive real-time feedback when making a mistake in the application regarding the bio-safety protocol.
(4) "Static Feedback" condition: Subjects use Kinect and can access a text manual upon request when getting stuck.

Note that we use a within-subject design when comparing Kinect to the mouse interface, and a between-subject design when comparing dynamic and static feedback conditions. Four subjects did the experiment in parallel. They are assigned to groups A,

[2]While we use textual feedback only, other types of feedback might be considered depending on the target group [Bouchard et al. 2012].

Table I. Group Distribution

A	B	C	D
Kinect	Kinect	Mouse	Mouse
Mouse	Mouse	Kinect	Kinect
Dynamic	Static	Dynamic	Static

B, C, and D, and encounter the conditions as shown in Table I. Our study well exceeds the number of required subjects for a usability study [Hwang and Salvendy 2010].

Subjects were told to perform a simple routine called "Isolation of infectious bacteria in human blood sample" in the virtual environment. While the subject was carrying out this task, the system automatically created a hazardous spill accident when the user opened the incubator, which was a mandatory action for the initial task. In accordance with the protocol, the "User Treats Spill" task was triggered.

5.1.2. Materials and Apparatus. When interacting with Kinect, subjects were standing 1.6m from a 42in monitor. When interacting with the mouse device, subjects were seated at a table in front of the monitor. Regarding subjects' subjective experience about usability, we prepared questions with answers in a Likert scale (from "1: Strongly disagree" to "5: Strongly agree"). The list of questions is composed of the following:

(1) Nine questions about the degree of usability of the application, employing a very extensively used questionnaire created by Brooke [1996] and two additional questions. The questions are shown in Table V.
(2) Three questions to obtain a measure of the usability of the gestural interface, and an additional two comparative questions. The questions are shown in Table VI.
(3) Seven application-specific questions about users' experience. The questions are shown in Table VII.

Subjects were also tested on their recall of the correct order of actions to be carried out of the spill treatment. There were seven multiple-choice questions with three multiple-choice options, such as "What is the correct spill treatment order?", "Which is the correct spill disposal procedure?", etc.

5.1.3. Procedure. The study was assisted by four nontechnical assistants, three technical assistants, and two experimenters, and was divided into four main parts:

(1) Welcome: Subjects are welcomed and receive a general introduction (7min)
(2) Tutorial and Usability Assessment: Subjects learn how to control the environment with Kinect (13min) and mouse and keyboard (10min) in tutorial style, followed by filling out usability questionnaires (Tables V and VI, 5min).
(3) Protocol Reading: Subjects are presented a protocol (standard guidelines) about the treatment of a toxic spill (5min).
(4) Spill Treatment and Knowledge Acquisition: Subjects solve the spill accident with the Kinect interface (20min). Afterwards, they first do the knowledge acquisition test (seven multiple-choice questions), then answer the questions from Table VII about the application experience (5min). Finally, they receive the reward.

The experiment lasted for 75min with a 5min break after 35min.

5.2. Results

5.2.1. Mistakes and System Feedback. The "Spill Treatment" task given to the subjects is subdivided into five ordered steps, or subtasks: (i) Bordering the spill, (ii) covering the spill, (iii) disinfecting the spill, (iv) merging the waste, and (v) disposing the waste. A sample error message shown to the user is seen in Figure 12.

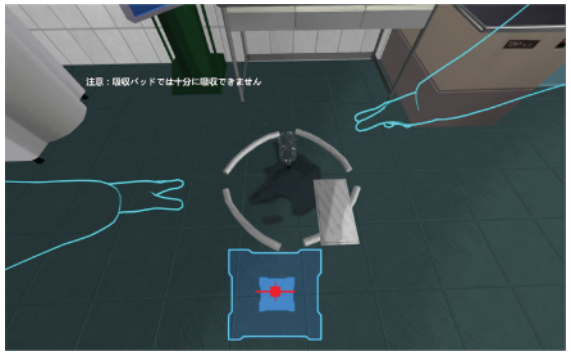

Fig. 12. Error message delivered by the system when the user attempts to cover the spill with an absorbent pad. Here, the user has already dropped a pad over the spill and is trying to drop a second one. The system promptly detected the intention of the user and displayed a warning message regarding the insufficiency of such action. The message in Japanese says: `It (the spill) cannot be sufficiently absorbed with an absorbent pad.`

Table II. Average Time (in Seconds) Consumed to Complete Each of the Five Steps

Condition	Step 1	Step 2	Step 3	Step 4	Step 5
Dynamic Feedback	197.3	308.8	218.7	135.4	52.6
Static Feedback	231.2	449.5	257.4	139.7	47.7

Table III. Average Time Consumed for Correcting Mistakes and Standard Deviation (In Seconds) in Dynamic and Static Feedback Groups

	Dynamic feedback		Static feedback	
Type of mistake	Average	Std.dev	Average	Std.dev
Interposing between spill and air vent	5.66	8.65	11.75	19.97
Cover spill with sponge	91.99	63.55	188.92	208.53
Cover spill with pad	43.44	45.23	271.20	199.20
Cover center of spill before outsides	22.34	24.97	54.30	61.31
Disinfect center of spill before outsides	0.35	0.07	NM	NM
Throw waste without merging	97.18	58.74	116.00	30.61
Throw waste in wrong bin	49.00	N/A	22.00	N/A

Note: "NM" abbreviates "no mistake" and N/A means that there was only one mistake; thus, standard deviation is not computed.

In the dynamic feedback version, subjects could complete 4.6 steps, on average, during the allotted 20min, with ten subjects completing all five steps. In the static feedback version, subjects could complete 3.3 steps, on average, with four subjects completing all five steps. This shows that, in the dynamic feedback version, more users were able to complete the spill treatment, and in general they completed more steps of the task. The average duration for the tasks is shown in Table II.

We now turn to the general analysis of user mistakes that occurred during the interaction with the system. Table III reports on the average time taken by subjects to correct one instance of a mistake in the dynamic-feedback version and static-feedback version, respectively, and Table VIII (see Online Appendix) shows the t-test results for comparing the two groups. The results indicate that dynamic feedback is significantly more effective than static feedback, that is, users receiving real-time assistance could recover from errors significantly faster. This finding supports our claim of the importance of real-time task recognition.

5.2.2. Usability and Usefulness. Figure 13 and Table IX (in Online Appendix) present the results of the usability of Kinect and mouse/keyboard. The usability results, specifically

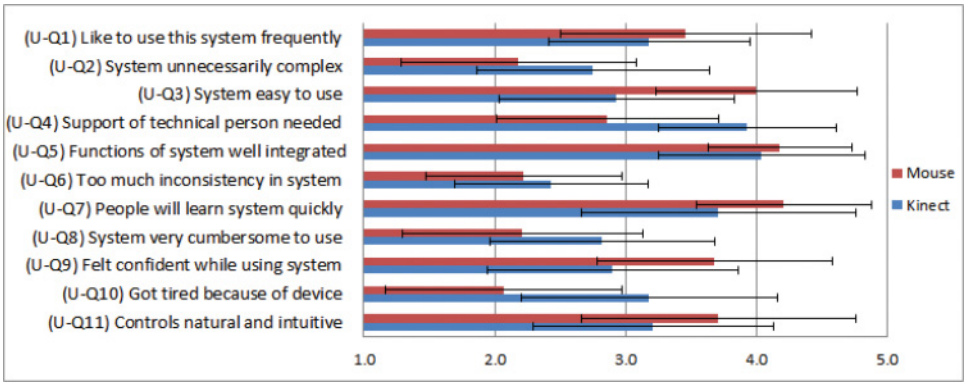

Fig. 13. Averages and standard deviation for usability questionnaire, as in Table V. "1: Strongly disagree"; "5: Strongly agree".

Table IV. Additional Questions on Kinect and Comparative Questions

Question stub	Average	Std.dev
(K-Q1) Moving across scenario intuitive	3.29	1.01
(K-Q2) Controlling camera viewpoint intuitive	3.04	1.07
(K-Q3) Taking and releasing objects intuitive	3.00	0.98
(C-Q1) Kinect more usable than mouse/keyboard	2.39	0.83
(C-Q2) Mouse/keyboard more usable than Kinect	4.11	0.63

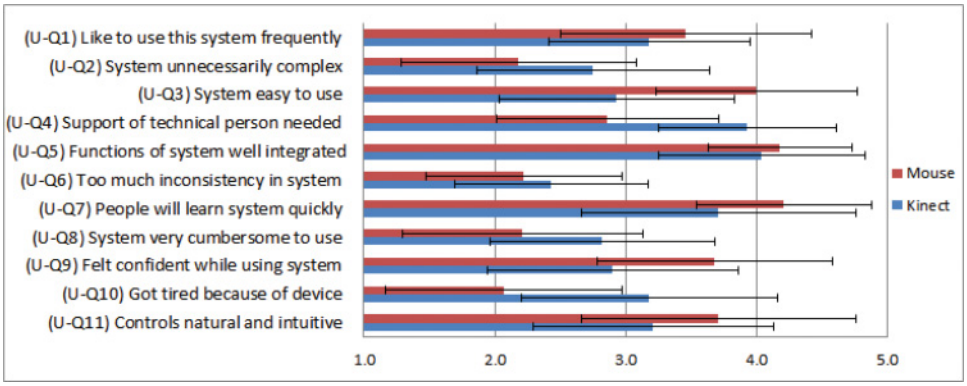

Fig. 14. Averages and standard deviation for application-specific experience questionnaire, comparing dynamic feedback to static feedback.

U-Q3 and U-Q10, indicate that the subjects preferred the mouse interface over the Kinect interface with statistical significance.

Table IV shows the results for the questions specific to Kinect and the comparative questions. Judging from the numbers, the perception of operating Kinect is neutral. Informal comments on usability of Kinect also provide some valuable insights. The advantage of Kinect is felt only after some practice time. Still, Kinect is seen as more tiring than the mouse interface in the free-format feedback as well; the mouse/keyboard device is seen as more usable than the Kinect interface.

Figure 14 presents the results on the application experience questions in both dynamic and static feedback groups. Table X (in Online Appendix) shows the t-test results for the two feedback groups.

It is important to note that the medical students considered the application as very useful for laboratory training procedures (Question A-Q7).

5.2.3. Knowledge Acquisition. The final question was whether dynamic feedback or static feedback is superior as to the recall of bio-safety lab protocols. Regarding the seven multiple-choice questions, in the dynamic version (14 subjects), there were 13.3 correct answers, on average; in the static version (14 subjects), there were 13.4 correct answers, on average. This indicates that the type of feedback has no specific impact on knowledge acquisition of the protocol. We speculate that the subjects remembered the protocol, which they read just before the spill-cleaning task, equally well. It is interesting to note that the students using the dynamic-feedback version had the impression of better recall of the session and better understanding of spill handling and mistakes (Questions A-Q1 to A-Q6).

6. CONCLUSIONS

In this article, we describe the implementation of an intelligent training or rehearsal system for biohazard laboratory procedures, based on the real-time instantiation of task models and mouse/keyboard input or gestural interaction. The primary contribution of the article is an integration of real-time task recognition, error-recognition strategies and error-message generation. This is achieved by providing adequate feedback messages when users make a mistake or do not know how to proceed. The secondary contribution is the integration of real-time task recognition with gestural interaction.

A field study demonstrated the robustness of the system and the usefulness of corrective feedback messages for fast recovery. Users would easily understand and heed the message when performing incorrect actions. It is important to note that errors are more easily described as part of a dynamic task representations than in a manual, as they are instanced only once in the task tree, whereas a manual would require repeating error descriptions in every context in which they can occur, increasing the amount of text for the user to read. For instance, subjects confused the absorbent paper with the absorbent pad because of their similar look. One subject (in the static-feedback condition) even completed the first and second stages of the spill handling with this wrong tool without noticing. In this way, we hope that users can enhance their knowledge of the training procedure by getting acquainted with the experience of "how not to do it."

A new finding from the research was that, quantitatively, the mouse and keyboard interface was clearly superior to the Kinect interface for our target user group, as judged from usability indicators. Still, the Kinect interface did not have negative ratings, and received some supportive comments in the free-text part of the questionnaire. We expect to improve the usability of the Kinect interface by some minor improvements of the application, such as preventing the undoing of some successful tasks. Gestures are convenient for use in virtual-reality environments such as a CAVE [Cabral et al. 2005]. Furthermore, a gesture-based immersive environment has also been shown to increase children's engagement with learning materials [Scarlatos and Friedman 2007].

In addition to the implementation of more scenarios, we want to extend the capabilities of the system by including a decision component (or scenario director) that may modify the environment to test and enhance users' understanding of the training scenario. For instance, the decision component could trigger a specific change in the environment (e.g., trigger another accident at a specific moment) to challenge users into dealing with two (or more) scenarios at the same time. Integrating our task recognition capabilities with a scenario director would allow us to create unexpected situations in which users might learn to apply their biohazard training knowledge in new and creative ways.

ACKNOWLEDGMENTS

The authors would like to thank A. Nakasone, F. Fonseca, and A. Rickett for their help with system development, and E. Gray for his help with the graphical assets.

REFERENCES

Ofra Amir and Ya'akov (Kobi) Gal. 2011. Plan recognition in virtual laboratories. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI'11)*. 2392–2397.

A. N. Angelov and Z. A. Styczynski. 2007. Computer-aided 3D virtual training in power system education. In *IEEE General Meeting of the Power Engineering Society*. IEEE, 1–4.

Chris L. Baker, Rebecca Saxe, and Joshua B. Tenenbaum. 2009. Action understanding as inverse planning. *Cognition* 113, 3, 329–349.

C. Barot, D. Lourdeaux, J. M. Burkhardt, K. Amokrane, and D. Lenne. 2013. V3s: A virtual environment for risk-management training based on human-activity models. *Presence: Teleoperators and Virtual Environments* 22, 1, 1–19. DOI: http://dx.doi.org/10.1162/PRES_a_00134

Olavo Da Rosa Belloc, Rodrigo B. D. Ferraz, Marcio Calixto Cabral, Roseli de Deus Lopes, and Marcelo Knörich Zuffo. 2012. Virtual reality procedure training simulators in X3D. In *Web3D*, Christophe Mouton, Jorge Posada, Yvonne Jung, and Marcio Cabral (Eds.). ACM, 153–160.

Francis Bisson, Hugo Larochelle, and Froduald Kabanza. 2015. Using a recursive neural network to learn an agent's decision model for plan recognition. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI'15)*, Buenos Aires, Argentina, July 25–31, 2015. 918–924. http://ijcai.org/papers15/Abstracts/IJCAI15-134.html.

B. Bouchard, F. Imbeault, A. Bouzouane, and B.-A. Menelas. 2012. Developing serious games specifically adapted to people suffering from Alzheimer. In *Proceedings of Serious Games Development and Applications*. Lecture Notes in Computer Science, Vol. 7528, Springer, Berlin, 243–254.

Doug A. Bowman and Larry F. Hodges. 1997. An evaluation of techniques for grabbing and manipulating remote objects in immersive virtual environments. In *Proceedings of the 1997 Symposium on Interactive 3D Graphics (I3D'97)*. ACM, New York, NY, 35–38. DOI: http://dx.doi.org/10.1145/253284.253301

Kirsty Bradbrook, Graham Winstanley, David Glasspool, John Fox, and Richard Griffiths. 2005. AI planning technology as a component of computerised clinical practice guidelines. In *Proceedings of the 10th Conference on Artificial Intelligence in Medicine (AIME'05)*. Springer, Berlin, 171–180. DOI: http://dx.doi.org/10.1007/11527770_26

J. Brooke. 1996. SUS: A quick and dirty usability scale. In *Usability Evaluation in Industry*, P. W. Jordan, B. Weerdmeester, A. Thomas, and I. L. Mclelland (Eds.). Taylor and Francis, Abingdon, Oxford, UK.

Hung Hai Bui, Svetha Venkatesh, and Geoff A. W. West. 2011. Policy recognition in the abstract hidden Markov model. *CoRR* abs/1106.0672 (2011). http://arxiv.org/abs/1106.0672

M. C. Cabral, C. H. Morimoto, and M. K. Zuffo. 2005. On the usability of gesture interfaces in virtual environments. In *Proceedings of the 2005 Latin American Conference on Human-Computer Interaction (CLIHC'05)*. ACM Press, 100–108.

Edward Carpenter, Induk Kim, Laura L. Arns, Mohan J. Dutta-Berman, and Krishna P. C. Madhavan. 2006. Developing a 3D simulated bio-terror crises communication training module. In *VRST*, Mel Slater, Yoshifumi Kitamura, Ayellet Tal, Angelos Amditis, and Yiorgos Chrysanthou (Eds.). ACM, 342–345.

Marc Cavazza, Simon Hartley, Lugrin, Jean-Luc, Bras, and Mikael Le. 2004. Qualitative physics in virtual environments. In *Proceedings of the 2004 International Conference on Intelligent User Interfaces (Virtual Environments & Stories)*. 54–61. http://doi.acm.org/10.1145/964442.964454

Marc Cavazza and Altion Simo. 2003. A virtual patient based on qualitative simulation. In *IUI'03*. ACM, 19–25. http://doi.acm.org/10.1145/604045.604053

Cristina Conati, Abigail S. Gertner, Kurt Vanlehn, and Marek J. Druzdzel. 1997. On-line student modeling for coached problem solving using Bayesian networks. In *User Modeling, Proceedings of the 6th International Conference (UM'97)*, Chia Laguna, Sardinia, Italy, June 2–5, 1997. Springer, 231–242.

Francesco Corato, Maria Frucci, and Gabriella Sanniti di Baja. 2012. Virtual training of surgery staff for hand washing procedure. In *AVI*, Genny Tortora, Stefano Levialdi, and Maurizio Tucci (Eds.). ACM, 274–277.

Albert Corbett, Megan McLaughlin, and K. Christine Scarpinatto. 2000. Modeling student knowledge: Cognitive tutors in high school and college. *User Modeling and User-Adapted Interaction* 10, 2–3, 81–108. DOI: http://dx.doi.org/10.1023/A:1026505626690

Ken Currie and Austin Tate. 1991. O-plan: The open planning architecture. *Artificial Intelligence* 52, 1, 49–86.

Kutluhan Erol, James A. Hendler, and Dana S. Nau. 1994. UMCP: A sound and complete procedure for hierarchical task-network planning. In *AIPS*. 249–254. http://www.informatik.uni-trier.de/~ley/db/conf/aips/aips1994.html#ErolHN94.

David Franklin, Jay Budzik, and Kristian J. Hammond. 2002. Plan-based interfaces: Keeping track of user tasks and acting to cooperate. In *IUI*. 79–86.

Ya'akov Gal, Swapna Reddy, Stuart M. Shieber, Andee Rubin, and Barbara J. Grosz. 2012. Plan recognition in exploratory domains. *Artificial Intelligence* 176, 1, 2270–2290.

Jennifer Gaudioso, Lisa Astuto Gribble, and Reynolds M. Salerno. 2009. Biosecurity: Progress and challenges. *Journal of the Association for Laboratory Automation* 14, 3, 141–147.

Christopher W. Geib and Robert P. Goldman. 2009. A probabilistic plan recognition algorithm based on plan tree grammars. *Artificial Intelligence* 173, 11, 1101–1132. DOI:http://dx.doi.org/10.1016/j.artint.2009.01.003

Gersende Georg and Marc Cavazza. 2007. Integrating document-based and knowledge-based models for clinical guidelines analysis. *Artificial Intelligence in Medicine* 421–430.

Andrew S. Gordon. 2003. Authoring branching storylines for training applications. In *Proceedings of the 6th International Conference on Learning Sciences (ICLS'04)*. 230–237.

Teresa Gutierrez, Jorge Rodriguez, Yaiza Velaz, Sara Casado, Angel Suescun Cruces, and Emilio J. Sanchez. 2010. IMA-VR: A multimodal virtual training system for skills transfer in industrial maintenance and assembly tasks. In *RO-MAN*, Carlo Alberto Avizzano and Emanuele Ruffaldi (Eds.). IEEE, 428–433. http://dblp.uni-trier.de/db/conf/ro-man/ro-man2010.html#GutierrezRVCCS10.

M. Hasanuzzaman, T. Zhang, V. Ampornaramveth, H. Gotoda, Y. Shirai, and H. Ueno. 2007. Adaptive visual gesture recognition for human-robot interaction using a knowledge-based software platform. *Robotics and Autonomous Systems* 55, 8, 643–657.

W. Hwang and G. Salvendy. 2010. Number of people required for usability evaluation: The 10 ± 2 rule. *Communications of the ACM* 53, 5, 130–133.

W. Lewis Johnson and Jeff Rickel. 1997. Steve: An animated pedagogical agent for procedural training in virtual environments. *SIGART Bulletin* 8, 1–4, 16–21.

Froduald Kabanza, Julien Filion, Abder Rezak Benaskeur, and Hengameh Irandoust. 2013. Controlling the hypothesis space in probabilistic plan recognition. In *IJCAI*, Francesca Rossi (Ed.). IJCAI/AAAI. http://dblp.uni-trier.de/db/conf/ijcai/ijcai2013.html#KabanzaFBI13.

H. A. Kautz and J. F. Allen. 1986. Generalized plan recognition. In *Proceedings of AAAI'86*. Philadelphia, PA, 32–37.

Per Ola Kristensson, Thomas Nicholson, and Aaron Quigley. 2012. Continuous recognition of one-handed and two-handed gestures using 3D full-body motion tracking sensors. In *Proceedings of the 2012 ACM International Conference on Intelligent User Interfaces (IUI'12)*. ACM, New York, NY, 89–92.

Florent Lalys, Laurent Riffaud, David Bouget, and Pierre Jannin. 2012. A framework for the recognition of high-level surgical tasks from video images for cataract surgeries. *IEEE Transactions on Biomedical Engineering* 59, 4, 966–976.

Marc Erich Latoschik and Christian Fröhlich. 2007. Towards intelligent VR - multi-layered semantic reflection for intelligent virtual environments. In *GRAPP (AS/IE)*. 249–260.

Gan Lu, Lik-Kwan Shark, Geoff Hall, and Ulrike Zeshan. 2012. Immersive manipulation of virtual objects through glove-based hand gesture interaction. *Virtual Reality* 16, 3, 243–252. DOI:http://dx.doi.org/10.1007/s10055-011-0195-9

Jean-Luc Lugrin and Marc Cavazza. 2007. Making sense of virtual environments: Action representation, grounding and common sense. In *IUI*, David N. Chin, Michelle X. Zhou, Tessa A. Lau, and Angel R. Puerta (Eds.). ACM, 225–234.

Héctor Muñoz-Avila, David W. Aha, Dana S. Nau, Rosina Weber, Len Breslow, and Fusun Yaman. 2001. SiN: Integrating case-based reasoning with task decomposition. In *IJCAI*. 999–1004.

Dana Nau, Malik Ghallab, and Paolo Traverso. 2004. *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers Inc., San Francisco, CA.

Dana S. Nau, Stephen J. J. Smith, and Kutluhan Erol. 1998. Control strategies in HTN planning: Theory versus practice. In *AAAI/IAAI*. 1127–1133.

Iason Oikonomidis, Nikolaos Kyriazis, and Antonis Argyros. 2011. Efficient model-based 3D tracking of hand articulations using Kinect. In *Proceedings of the British Machine Vision Conference*. BMVA Press, 101.1–101.11.

David V. Pynadath and Michael P. Wellman. 2000. Probabilistic state-dependent grammars for plan recognition. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI'00)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, 507–514. http://dl.acm.org/citation.cfm?id=2073946.2074005

M. Ramirez and H. Geffner. 2009. Plan recognition as planning. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI'09)*.

M. Ramirez and H. Geffner. 2010. Probabilistic plan recognition using off-the-shelf classical planners. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI'10)*.

Jason E. Rao. 2011. United States Department of State's biosecurity engagement program: Bio threat reduction through international partnerships. In *Encyclopedia of Bioterrorism Defense*, Washington, DC.

L. Scarlatos and R. Friedman. 2007. *On Developing User Interfaces for Children in Educational Virtual Reality Systems*. Technical Report. CUNY Graduate Center Technical Report TR-2007001, New York, NY.

Michael Schneider. 2010. *Resource-aware Plan Recognition in Instrumented Environments*. Ph.D. Dissertation. Saarland University, Saarbruecken.

Yuval Shahar, Silvia Miksch, and Peter D. Johnson. 1998. The Asgaard project: A task-specific framework for the application and critiquing of time-oriented clinical guidelines. *Artificial Intelligence in Medicine* 14, 1–2, 29–51.

Yale Song, David Demirdjian, and Randall Davis. 2012. Continuous body and hand gesture recognition for natural human-computer interaction. *ACM Transactions on Interactive Intelligent Systems* 2, 1, 1–28.

Young Chol Song, Henry Kautz, James Allen, Mary Swift, Yuncheng Li, Jiebo Luo, and Ce Zhang. 2013. A Markov logic framework for recognizing complex events from multimodal data. In *Proceedings of the 15th ACM on International Conference on Multimodal Interaction (ICMI'13)*. ACM, New York, NY, 141–148. DOI: http://dx.doi.org/10.1145/2522848.2522883

Anuraag Sridhar and Arcot Sowmya. 2008. Multiple camera, multiple person tracking with pointing gesture recognition in immersive environments. In *Advances in Visual Computing*, George Bebis, Richard Boyle, Bahram Parvin, Darko Koracin, Paolo Remagnino, Fatih Porikli, Jrg Peters, James Klosowski, Laura Arns, YuKa Chun, Theresa-Marie Rhyne, and Laura Monroe (Eds.). Lecture Notes in Computer Science, Vol. 5358. Springer, Berlin, 508–519. DOI: http://dx.doi.org/10.1007/978-3-540-89639-5_49

Catherine Stocker, Benjamin Sunshine-Hill, John Drake, Ian Perera, Jr. Joseph T. Kider, and Norman I. Badler. 2011. CRAM it! A comparison of virtual, live-action and written training systems for preparing personnel to work in hazardous environments. 95–102.

Gita Sukthankar, Robert P. Goldman, Christopher Geib, David V. Pynadath, and Hung Hai Bui. 2014. An introduction to plan, activity, and intent recognition. In *Plan, Activity, and Intent Recognition: Theory and Practice*, Gita Sukthankar, Christopher Geib, Hung Hai Bui, David Pynadath, and Robert P. Goldman (Eds.). Morgan Kaufmann, Burlington, MA.

Gabriel Synnaeve and Pierre Bessière. 2011. A Bayesian model for plan recognition in RTS games applied to Starcraft. *CoRR* abs/1111.3735 (2011). http://arxiv.org/abs/1111.3735

Karim A. Tahboub. 2006. Intelligent human-machine interaction based on dynamic Bayesian networks probabilistic intention recognition. *Journal of Intelligent and Robotic Systems* 45, 1, 31–52.

Luis Unzueta, Oscar Mena, Basilio Sierra, and Angel Suescun. 2008. Kinetic pseudo-energy history for human dynamic gestures recognition. In *AMDO'08*, Lecture Notes in Computer Science, Francisco J. Perales Lopez and Robert B. Fisher (Eds.), Vol. 5098. Springer, Berlin, 390–399. http://dblp.uni-trier.de/db/conf/amdo/amdo2008.html#UnzuetaMSS08.

Etienne van Wyk and Ruth de Villiers. 2009. Virtual reality training applications for the mining industry. In *Afrigraph*, Alexandre Hardy, Patrick Marais, Stephen N. Spencer, James E. Gain, and Wolfgang Straßer (Eds.). ACM, 53–63.

Kurt Vanlehn, Collin Lynch, Kay Schulze, Joel A. Shapiro, Robert Shelby, Linwood Taylor, Don Treacy, Anders Weinstein, and Mary Wintersgill. 2005. The Andes physics tutoring system: Lessons learned. *International Journal of Artificial Intelligence in Education* 15, 3, 147–204. http://dl.acm.org/citation.cfm?id=1434930.1434932

Alberto Cabas Vidani and Luca Chittaro. 2009. Using a task modeling formalism in the design of serious games for emergency medical procedures. In *Conference in Games and Virtual Worlds for Serious Applications, VS-GAMES'09*, Coventry, UK, March 23–24, 2009, Genaro Rebolledo-Mendez, Fotis Liarokapis, and Sara de Freitas (Eds.). IEEE Computer Society, 95–102.

Dennis Wiebusch and Marc Erich Latoschik. 2012. Enhanced decoupling of components in intelligent real-time interactive systems using ontologies. In *SEARIS*. 43–51.

David Wilkins and Marie desJardins. 2001. A call for knowledge-based planning. *AI Magazine* 22, 1, 99–115.

Received June 2015; revised March 2016; accepted May 2016