Center for Human Modeling and Simulation

Department of Computer & Information Science

February 2007

Automated Analysis of Human Factors Requirements

Jan Allbeck
University of Pennsylvania, allbeck@seas.upenn.edu

Norman I. Badler
University of Pennsylvania, badler@seas.upenn.edu

Recommended Citation

Allbeck, J., & Badler, N. I. (2007). Automated Analysis of Human Factors Requirements. Retrieved from http://repository.upenn.edu/hms/4

Copyright 2006 SAE International.
Postprint version. Published in *Proceedings of the 2006 Digital Human Modeling for Design and Engineering Conference,* July 2006, Document 2001-01-2366, 7 pages.
Publisher URL: http://www.sae.org/technical/papers/2006-01-2366

Automated Analysis of Human Factors Requirements

Abstract

Computational ergonomic analyses are often laboriously tested one task at a time. As digital human models improve, we can partially automate the entire analysis process of checking human factors requirements or regulations against a given design. We are extending our Parameterized Action Representation (PAR) to store requirements and its execution system to drive human models through required tasks. Databases of actions, objects, regulations, and digital humans are instantiated into PARs and executed by analyzers that simulate the actions on digital humans and monitor the actions to report successes and failures. These extensions will allow quantitative but localized design assessment relative to specific human factors requirements.

Keywords

digital humans

Comments

Postprint version. Published in *Proceedings of the 2006 Digital Human Modeling for Design and Engineering Conference*, July 2006, Document 2001-01-2366, 7 pages.
Publisher URL: http://www.sae.org/technical/papers/2006-01-2366

06DHM-49

Automated Analysis of Human Factors Requirements

Jan M. Allbeck and Norman I. Badler
University of Pennsylvania

ABSTRACT

Computational ergonomic analyses are often laboriously tested one task at a time. As digital human models improve, we can partially automate the entire analysis process of checking human factors requirements or regulations against a given design. We are extending our Parameterized Action Representation (PAR) to store requirements and its execution system to drive human models through required tasks. Databases of actions, objects, regulations, and digital humans are instantiated into PARs and executed by analyzers that simulate the actions on digital humans and monitor the actions to report successes and failures. These extensions will allow quantitative but localized design assessment relative to specific human factors requirements

INTRODUCTION

Recent improvements in computation speed and control methods have allowed the portrayal of 3D digital humans suitable for interactive, real-time applications. Manually controlling them, however, is an undesirable burden to many designers and a bottleneck in evaluating a large-scale designed environment such as a ship. We have developed a Parameterized Action Representation (PAR) that dramatically increases access to the functionality of digital humans through a simpler, task-oriented interface. The PAR specifies the agent of the action, any relevant objects, and essential action information concerning path, location, manner, purpose and termination conditions.

Typically, textual task requirements or instructions lack enough information to uniquely visualize a digital human's performance. PAR includes information to fill in these gaps. The object representation includes semantic tags that connect human manipulations to actions that the object can perform and what state changes they cause to all participants. The action is spatially situated since the approach to the object, the path taken, and poses and the arm reaches needed are affected by the agent's size, starting location, and obstacles. PARs may be stored in a re-usable and extensible hierarchical database called an Actionary. Building a database of parameterized actions and semantically tagged virtual objects, and connecting the PAR simulation system to them will allow CAD designs to be automatically tested against a human model population who virtually perform the required operational tasks. PARs can check a design against prewritten regulations, such as accessibility, clearance, and operability. Such requirements can be represented as instructions or predicates and stored in a database; they can be iteratively executed against a range of agents and object configurations. Execution failures can be summarized and reported. Visual simulations and analyses can also alert designers to areas of congestion or inefficient layout. Using this tool to help automate as well as encourage iterative engineering design will help prevent costly hardware modifications or awkward task corrections. Creating separable databases of requirements and actions may encourage the digital human modeling community to consolidate and incorporate analysis capabilities and features through high level programming interfaces.

In this paper we will give a short overview of our Parameterized Action Representation and its software system. We will also discuss extensions to our previous work and how they can be used to automate requirements checking. We will then present an example and conclude with a summary and other possible extensions and applications.

PARAMETERIZED ACTION REPRESENTATION

Our Parameterized Action Representation (PAR) was designed as an intermediary between natural language and animation. A software system for interpreting PARs and animating them was designed and implemented. Previous applications of the PAR software system focused on the creation of virtual environments for training [6]. The details of PAR and the PAR software system can be found in previous publications [2, 3, 6, 7]. In this paper we will provide only a quick overview of the representation and components and will focus on developments needed to successfully use the PAR framework for automatic testing of regulations.

ACTION AND OBJECT HIERARCHIES

PARs are stored in two hierarchical databases: an action hierarchy called the Actionary and an object hierarchy. The hierarchical nature of the databases facilitates the addition of new actions and objects. Once the databases are populated with base actions and objects, new entries can be added by finding their proper placement in the hierarchy and simply specifying their distinguishing parameters. Inheritance will fill in all of the other parameters. For example, *fasteners* may be specified high in the object hierarchy. *Bolts* would then be a child of *fasteners* inheriting properties such as the object's *purpose* to connect other objects. The *bolt* entry would additionally specify tools that can operate on it (*e.g. wrenches*) and actions that can be applied to it (*e.g., inserting, removing*). A *bolt* would also include parameters for *tightening* and *loosening* and be linked to *threaded holes*, *nuts* and *washers*.

PARs come in two forms, uPARs and iPARs. uPARs are uninstantiated PARs, lacking characteristics specific to a particular scenario; think of them as *patterns*. iPARs are uPARs that have been instantiated. For example, a *bolt* might have its *size* parameters instantiated with 3/4" head and 5/16" thread. An action, such as *tightening*, should be instantiated with the digital human performing the action and the specific object(s) being tightened.

KEY FIELDS OF THE ACTION REPRESENTATION

Participants are the agent and object parameters of PARs. The agent is the digital human executing the action. For regulation testing, selected digital humans from some population can be assigned as agents. The object type is defined explicitly for a complete representation of a physical object and is stored hierarchically in a database. Each object in the environment is an instance of this type and is associated with a graphical model in a scene graph.

Some of the fields of PAR are designed to aid in or short-cut the task planning process. The applicability conditions of an action specify what needs to be true in the world in order to carry out an action. These can refer to agent capabilities, object configurations, and other unchangeable or uncontrollable aspects of the environment. The conditions in this boolean expression must be true to perform the action. For *walk*, one of the applicability conditions may be: *Can the agent walk?* If these conditions are not satisfied, the action cannot be executed.

Preparatory specifications are a list of <condition, action> statements. The conditions are evaluated first and have to be satisfied before the current action can proceed. If the conditions are not satisfied, then the corresponding action is executed; it may be a single action or a complex PAR. In general, preparatory specifications produce a limited form of automated planning; e.g., to indicate explicitly that a handle has to be grasped before it can be turned in order to open a panel.

Termination conditions are a list of conditions that, when satisfied, complete the action. Terminations may be due to success or failure; these conditions are distinguished as they are critical to a task analysis. Particularly in applications dealing with mechanical devices, termination conditions can be crucial. Actions such *loosening* and *removing* a nut would result in similar performances, but with one terminating before the nut falls from the bolt and the other when the nut falls from the bolt. This may be the difference between successfully completing a maintenance task and needing to completely rebuild a device.

OBJECT REPRESENTATION ENHANCEMENTS

To accomplish our simulation goals, we must enhance the information content in the objects with which the digital humans interact. We expect to use some portions of a Product Data Management (PDM) system [16-18] so that objects may possess and express information essential to their function and use.

Existing components of the PDM include: properties (*e.g. steel*, *wood*), status (e.g. *broken*, *idle*, *operating*), posture (e.g. *open*, *closed*, *tight*, *loose*), location (e.g. part of *room2*, part of *pump3*), contents (i.e. parts of the object), relative directions (e.g. *top*, *bottom*, *front*, *back*), and lower level graphics data such as bounding volumes, coordinate system, position, orientation, velocity, and sites (oriented points). Each object is also associated with a graphical CAD model, currently linked through a pointer to the corresponding object in the *Jack* scene graph [13]. Note that much of this information is or could be generated well in advance of any ergonomics analysis.

We are adding parameters to the PDM to aid human factors requirements analyses. These include both high-level parameters for determining which regulations are relevant to the objects and low-level parameters that can be used by digital human motion generators (analyzers) to evaluate the regulations. Since the PDM organization may mirror the object assembly hierarchy rather than a functional hierarchy, we can pre-process the PDM for a specific design to build additional relations and tables. For example, one can construct a hierarchy based on object classification: e.g., all display objects would be found in the sub-tree of displays and all control actuators would be found in the sub-tree of actuators. In the representation of regulations this allows us to specify entire sub-trees instead of individual objects. Additional parameters in the object representation are also required to determine which requirements are applicable. For example, different distance regulations are specified for different frequencies of use, levels of precision, and for emergency controls. Also, controls should be grouped by function and distributed such that an operator is not overworking one end-effector while not utilizing another. An object's function can easily be stored in its PDM. For example, a toggle switch may contain its function: *operate(pump1)*.

Creating sophisticated analyzers and geometry checkers that can test requirements must include object parameters such as parts, purposes, access paths, tools, default state, actions that can be performed on the objects and actions that can be performed with the objects. Many of these parameters can be specified in the root of an object sub-tree and inherited by the children. Some will need to be user- or designer-specified, while others may be filled in by automated preprocessors operating directly on the CAD model. For example, walkable access paths may be automatically generated and stored in the object representation. In a ship design many parts such as pumps, consoles, and panels are used in many places and even reused in other ship designs possibly with minimal alterations. The specification of useful parameters in the PDM will allow the information to be re-used with minimum or no further user input.

Previously, we considered agents to be a special kind of object and stored them a separate sub-tree in the object hierarchy. For this particular application, we are not focusing on the representation of agents (digital humans) except for their anthropometry which can easily be stored as a table in the database. Future applications, such as efficient manning evaluations, will be aided by the inclusion of skills and roles in the agent representation.

REQUIREMENTS REPRESENTATION

Requirements can be specified by linking actions with object sub-trees. In looking at the *ABS Guidance Notes on the Application of Ergonomics to Marine Systems* [1], we found that many of the requirements deal with the spacing of objects and their parts. A requirement specifying the proper spacing of toggle switches can be written as follows:

spacing(ToggleSwitches,[table2])

Where *spacing* is a PAR, *ToggleSwitches* is a sub-tree of the object hierarchy, and *table2* is a reference table name specifying the required spacing. Note, that we do not envision spacing requirements to be altered often, so the table parameter is optional and may be specified if different spacing requirements are desired for a design. Otherwise, a default spacing table would be referenced. Note also that entire sub-tree can be specified, so checking the spacing of all control actuators could be specified by simply replacing *ToggleSwitches* with *ControlActuators* or, if only a portion of the toggle switches needed to be checked, they could be individually specified. This affords users considerable control with minimum effort.

This is a particularly simple example and does not fully illustrate the power of our representations. Another form of requirements checking would be to ensure that all necessary maintenance procedures can be successfully performed on the design. The following is taken from a NASA document for instructing the maintenance of a piece of exercise equipment:

Raise Lift Bar until Lift Bar clears the Upper Stops, allow the Lift Bar to come to rest on the Upper Stops.

This maintenance instruction would result in a complex iPAR with two execution steps: one for lifting the lift bar and another for setting the bar on the upper stops:

lift(dh1, liftBart, clear(liftBar1, upperStops1))

set(dh1, liftBar1, upperStops1, rests(liftBar1, upperStops1)),

where *dh1* is the digital human performing the actions, *liftBar1 and upperStops1* refer to objects found in the object database, and the last parameters represent the termination conditions. These requirements would then be assigned to individuals representing a desired population to determine the range of accommodation that could lift the bar high enough to meet the termination conditions of the actions. The next section will discuss how these evaluations are performed and how failures are reported.

FRAMEWORK FOR REQUIREMENTS TESTING

Figure 2 illustrates our framework and its components for testing requirements. We demonstrate how a design and ergonomic requirements would be processed.

A user would start with a CAD model of the design. There are well established avenues for converting CAD models from many standard systems into geometry compatible with digital human modeling systems. Semantic information about the objects and their components might be available through a PDM, entered by the user, generated automatically through inheritance, or created by software procedures that detect geometric features. Figure 1 shows an example object hierarchy. The leaf nodes correspond to actual geometric models in the design. The other nodes are created to be reused elsewhere and to facilitate inherence and thereby reduce data entry. For example, another “standard” toggle switch could be added simply by providing its name. Other critical semantic information would be inherited from the parent node. As with most inheritance schemes, inherited parameters can be overwritten in the child if desired.

Table 1 shows a few of the parameters for the object representation of a continuous rotary controller. The object’s name both in the object database and the scene graph is *rotary1*. It has two properties in its properties list: its material, *steel*, and the level of control required for operation, *precise*. Its status is *operational*, meaning that it is a functioning device not currently being used. Its posture is 5% open. It is located on *console1*. It has no specified contents (parts). The top of the device is specified by a site named *topSite*, which can be found in the scene graph. To increase the device's control function, it can be rotated clockwise. The purpose of the device is to control *valve1*. The only action that can be applied to the device is *rotate*, and the controller is frequently used relative to other controllers.

Name	rotary1
Properties	Steel, Precise
Status	Operational
Posture	5%
Location	console1
Contents	none
Relative directions	top(rotary1.topSite)
Functional directions	increase(clockwise)
Purpose	control(valve1)
Actions	rotate
Frequency (of use)	8 (out of 10)

Table 1: Example of object parameters and values.

A database of anthropometrically scaled digital humans can be generated and stored. We are currently using 5th, 50th, and 95th percentile male and female models [9]. The user may select which humans (all or a subset) to use for requirements checking.

The Actionary will already be populated with many of the actions required. Users will also be able to add additional actions by using existing actions as templates; the inheritance structure helps lessen the amount of data entry necessary.

Similarly, many requirements will already exist in the requirements database, allowing a user to re-use them directly or as templates for constructing new requirements. From the database of requirements a user will be able to specify which requirements are applicable to a given scenario.

The instantiator links the requirements, actions, objects, and digital humans and generates iPARs. During this process, simple checks will be done to ensure that the necessary objects can be found in the scene graph and the specified actions can be applied to those objects.

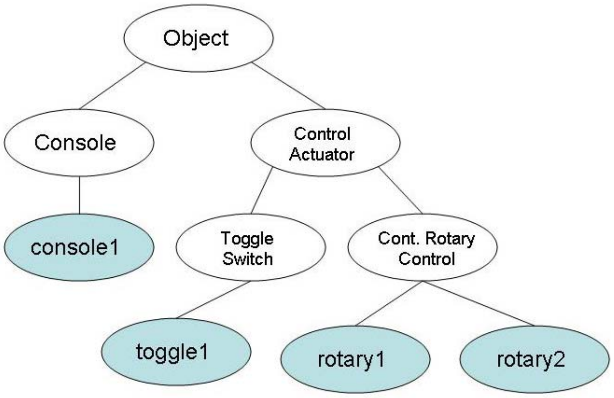

Figure 1: Example object hierarchy for ABS guidelines.

Checking for required tags will also take place (*e.g.*rotary1.topSite is in the scene graph). In our example, one of the iPARs generated as requirements could be: *spacing(Control_Actuator)*. This will check for the proper spacing requirements for all control actuators. Another requirement might be *rotate(dh1, rotary1, clockwise)*, which would result in digital human *dh1* performing a rotation of continuous rotary actuator *rotary1*. Note, that these are extremely simplified expressions of actual iPARs; iPARs are actually stored in a MYSQL database with many additional parameters.

After the iPARs are instantiated by the Instantiator, they are sent to an Agent Process. Each digital human in the scene graph is linked to an Agent Process. There will also be an additional Agent Process for processing geometric or spacing requirements not requiring actual digital human performance. Once in an Agent Process, the iPARs are processed for execution. For applications such as regulation checking, ordering of actions need not be specified. An action queue holds the iPARs to be performed and a process manager checks the conditions of an action to determine whether or not it can be performed in the current state of the agent and world [7]. For example, in order for *dh1* to rotate *rotary1*, he needs to grasp it. In order to grasp it, he needs to reach it. In order to reach it he needs to be within reaching distance. In order to be within reaching distance, he may need to walk and get into position. This pseudo-planning process is conducted by the agent process using the PAR parameters [7]. During this process the current state of the world must be checked. For example, is *dh1* grasping *rotary1*? To check this condition, the Agent Process will send a logic expression such as, *grasping(dh1, rotary1)* to the Predicate Manager. The Predicate Manager then consults the current state of the world as stored in World Model. If a geometric or spatial check is required, such as *reachable(dh1, rotary1)*, the Predicate Manager will call the Geometry/Spatial Checker which will do the necessary calculations based on information from the Scene Graph.

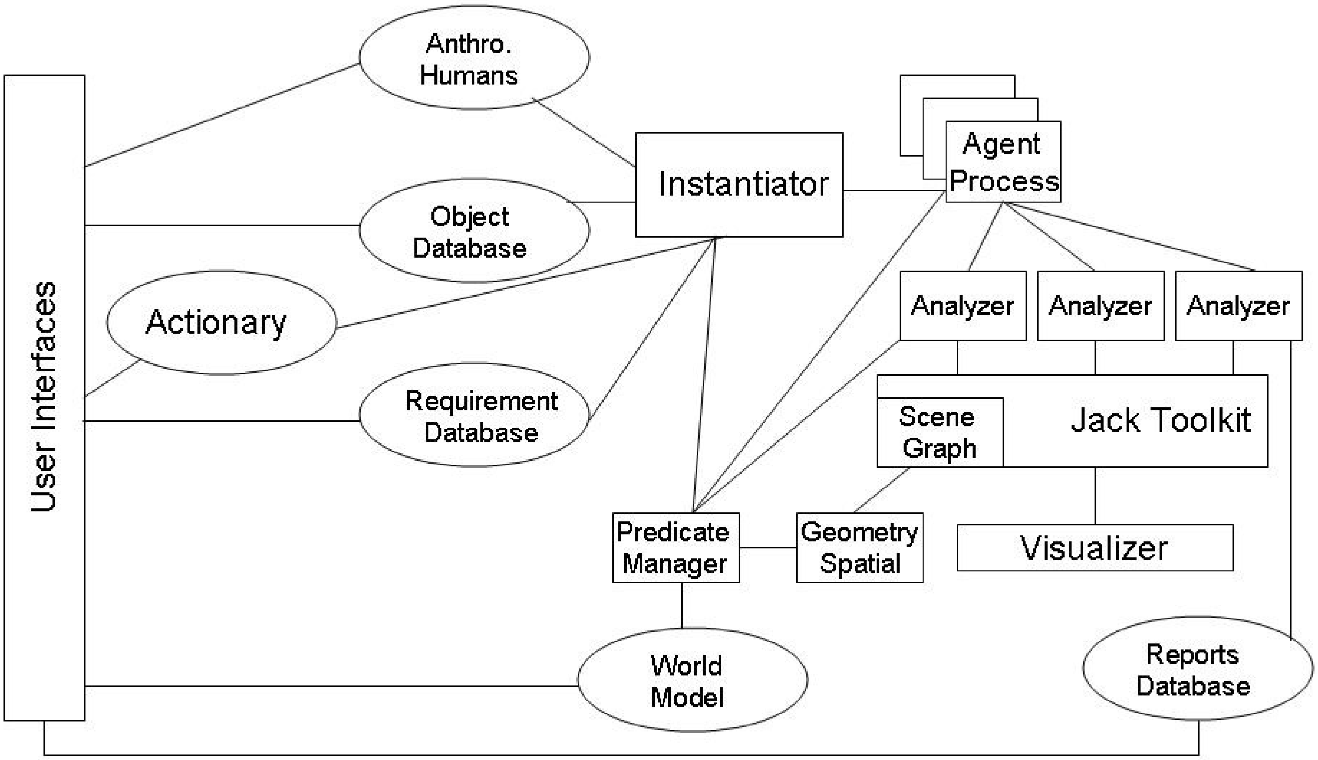

Figure 2: PAR system diagram

The pseudo-planning process often results in additional PARs being added to the agent's action queue. These actions are then processed in the same fashion. Once an action meets requirements for execution, the iPAR is passed by the agent to the registered Analyzer for that action.

Analyzers are generally procedural animation routines. They use the parameters specified in the iPAR and associated PDM to perform the action using the specified digital human. Action execution includes determining when an action should terminate. As outlined earlier, termination conditions are a PAR parameter. Analyzers can use the Predicate Manager to check for these conditions. For example, rotating *rotary1* may first require the execution of a *walk* PAR to get within reaching distance of *rotary1.* The *walk* PAR should terminate when *dh1* is within reaching distance. Our locomotion Analyzer queries the Predicate Manager with *distance(dh1, rotary1, reach)*. The Predicate Manager in turn will ask the Geometry/Spatial Checker to calculate the distance between *dh1* and *rotary1* and then compare it to the stored reach distance in the object representation of *dh1,* and respond to the Analyzer.

In addition to performing the animations, Analyzers are responsible for updating the World Model and reporting failures. The Analyzers update the World Model basically by informing the Predicate Manager of what actions are being performed and on which objects. For example, when the rotation of *rotary1* is being performed, it will send *rotating(dh1, rotary1*). Additionally, it will send updates for other parameters of the objects (e.g. *posture(rotary1, 10)*).

If an Analyzer cannot successfully complete an action, it will report an error to the Reports Database. See [3] for previously reported research on detecting errors. We will be iteratively constructing a format for the Analyzers to report errors such that it is readable by users. Currently failure reports include the iPAR that failed, a pointer to the previous iPAR executed and the parent iPAR if it is a complex iPAR, and a reason for the failure chosen from a set of previously constructed failure conditions. For example, the locomotion Analyzer may fail to get *dh1* within reaching distance of *rotary1*. It may then report the *walk(dh1, rotary1)* iPAR as failing with reason *no_path_found*. From the other parameters of the iPAR it would be determined that this iPAR was generated from the preparatory specifications of the *rotate(dh1, rotary1)* iPAR that was generated by the Instantiator and that requirement would be marked as failing. Additionally, upon failure, Analyzers report the failure to the issuing Agent Process and update the Predicate Manager in order to maintain a stable system allowing other (or alternative) iPARs to be executed.

Analyzers can animate objects as well as digital humans. For requirements checking they must also perform calculations. In our example, one of the requirements is to check for the proper spacing of all control actuators: *spacing(Control_Actuator).* This does not involve a digital human animation. It requires only distance checks. An analyzer can be constructed just for this purpose. According to [1], different control actuators require different spacing. Spacing requirements are given as tables in their document. These tables can be stored in the Analyzer and referenced during calculations. In our example, the spacing Analyzer would reference the object database for a list of all of the control actuators and begin calculating distances between them by obtaining their global locations from the scene graph. It would then compare the calculated distances against the distances in the table. For example, continuous rotary controls are to be placed 19mm from toggle switches. Errors will be reported just as with the Analyzers driving digital humans.

Many requirements may specify ranges of preference for locations of devices. For example, there are preferred ranges of locations for actuators. In this case there may be clear failures as well as warnings that would arise when the actuator is within a secondary range, but not the preferred range.

As the Analyzers report failures and warnings, they accumulate in a Reports Database. A user interface to the Reports Database enables viewing the reports as well as sorting or searching them.

Currently our Analyzers build on the functionality of the UGS Jack Toolkit [13] for a scene graph and for animating the digital humans and objects. The toolkit is connected to OpenGL for visualization [5].

CONCLUSION

Our goal for these action and object representations and the accompanying system to interpret them is to provide designers with an automated system for checking requirements against designs. Although other action representations have been developed, including [4, 8, 11, 15, 19], for the most part they have been applied to social agents, not digital humans for human factors analysis. Ianni has been working on creating an action specification for human factors analysis for some time [10, 11]. As he points out, if there is a strong enough linkage between the digital humans (bodies) and process models (minds), physical and non-physical aspects of tasks could be evaluated concurrently. For the most part, human factors analysis entails creating analysis tools or simulators for a particular concern (e.g. visibility, ingress and egress, reaching and grasping, strength assessment, etc). Each analysis must be crafted, set up, and run for each design being analyzed. In this paper we have presented a framework that will facilitate a range of testing on multiple designs. Such a system would simplify the creation of simulations for analysis while still providing detailed human simulation.

Other systems have been created that start to address a more universal analysis approach. Micro Analysis and Design has software systems that analyze timing of many different tasks, but they do not provide lower level analysis such as reachability [12]. Instruction agents like Steve [14] have a model of the environment and semantics of actions, but are not designed for human factors analysis. Their human simulation system does not provide the tools necessary for such analysis.

Ideally, there would be no user intervention needed other than picking a model, anthropometric population, and list of requirements. We are not there yet, but we believe this framework is extensible and that the Actionary, object (PDM) and requirements databases will need less and less data entry from users as they are populated.

We also believe that by using existing entries in the databases as templates and well-designed user interfaces, designers and other users without programming backgrounds will be able to use this system quickly and effectively. Furthermore, once the databases are constructed for an application, multiple designs or incremental design iterations can be readily checked for requirements adherence.

There are several extensions possible. The most pressing for this application is the continued construction of geometry checkers. Geometry checkers examine the “polygon soup” that often results from the conversion of CAD models into 3D models used in human factors analysis, and recognizes semantic information such as holes, handles, tops, connectors, joints, access paths, and the relationships between the parts. If more semantic information can be gathered automatically, then less data will have to be authored manually. PDMs are extremely helpful, but presently do not appear to contain all of the information required for effective human factors analyses.

Another extension is the inclusion of more agent properties with an application to manning evaluation. By representing an agent’s skill level for various tasks (via PARs) and their roles (jobs), entire agent crews could be asked to perform their duties. They could report back failures and timing information. Users (or computation) could observe overcrowded spaces in the design, and experiments could assess different manning configurations by assigning different numbers of agents and agents with differing skill levels and roles.

The creation and management of teams of agents is another possible extension that is beyond the scope of our current project. PARs could be assigned to teams of agents as easily as to individual agents, but a team manager would be required to coordinate the agents and resolve any conflicts.

ACKNOWLEDGMENTS

This research was partially supported by grants NSF IIS-0200983, NASA NRA 3-0BPR-01, Air Force FA8650-05-2-6649 and Lockheed Martin. We also gratefully acknowledge the support of Apple, Autodesk, NVIDIA, PIXAR, and Tecnomatix.

REFERENCES

1. ABS *Guidance Notes on the Application of Ergonomics to Marine Systems*. American Bureau of Shipping, New York, NY, 1998.
2. Allbeck, J. and Badler, N. Representing and Parameterizing Agent Behaviors. in Prendinger, H. and Ishizuka, M. eds. *Life-like Characters: Tools, Affective Functions and Applications*, Springer, Germany, 2003.
3. Allbeck, J., Bindiganavale, R., Kipper, K., Moore, M., Schuler, W., Badler, N., Joshi, A.K. and Palmer, M., Authoring Embodied Agents' Behaviors through Natural Language and Planning. in *Workshop on Key Problems for Creating Real-time Embodied Autonomous Agents at Autonomous Agents Conference*, (Barcelona, Spain, 2000).
4. Arafa, Y., Kamyab, K. and Mamdani, E., Character animation scripting languages: a comparison. in *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, (Melbourne, Australia, 2003), ACM Press, 920-921.
5. Badler, N., Allbeck, J., Lee, S.J., Rabbitz, R.J., Broderick, T.T. and Mulken, K.M., New Behavioral Paradigms for Virtual Human Models. in *SAE International Digital Human Modeling for Design and Engineering*, (2005).
6. Badler, N.I., Bindiganavale, R., Allbeck, J., Schuler, W., Zhao, L. and Palmer, M. Parameterized Action Representation for Virtual Human Agents. in Cassell, J. ed. *Embodied Conversational Agents*, MIT Press, 2000, 256-284.
7. Bindiganavale, R., Schuler, W., Allbeck, J.M., Badler, N.I., Joshi, A.K. and Palmer, M., Dynamically Altering Agent Behaviors Using Natural Language Instructions. in *Autonomous Agents*, (2000), 293-300.
8. Coyne, B. and Sproat, R., WordsEye: An Automatic Text-to-Scene Conversion System. in *SIGGRAPH 2001: Computer Graphics Proceedings*, (2001), ACM Press, 487-496.
9. Gordon, C.C. Anthropometric Survey of U.S. Army Personnel: Methods and Summary Statistics, 1988.
10. Ianni, J., Standardizing Human Model Queries. in *Society of Automotive Engineers (SAE)*, (2001).
11. Ianni, J.D., A Specification for Human Action Representation. in *Society of Automotive Engineers (SAE) Digital Human Modeling International Exposition Proceedings*, (1999).
12. MA&D. Micro Analysis & Design: IMPRINT, http://www.maad.com/index.pl/crew_station_design_tool, Last visited April 2006.
13. Raschke, U. UGS Jack, http://www.ugs.com/products/tecnomatix/human_performance/jack/, Visited Dec. 2005.
14. Rickel, J. and Johnson, W.L. Animated Agents for Procedural Training in Virtual Reality: Perception, Cognition, and Motor Control. *Applied Artificial Intelligence*, *13*. 343-382.
15. Swartout, W., Hill, R., Gratch, J., Johnson, W.L., Kyriakakis, C., LaBore, C., Lindheim, R., Marsella, S., Miraglia, D., Moore, B., J., M., Rickel, J., Thiébaux, M., Tuch, L., Whitney, R. and Douglas, J., Towards the Holodeck: Integrating Graphics, Sound Character and Story. in *Autonomous Agents*, (2001), 409--416.
16. Szykman, S., Sriram, R.D., Bochenek, C., Racz, J.W. and Senfaute, J. Design Repositories: Next-Generation Engineering Design Database. *IEEE Intelligent Systems*.
17. UGS. Open Product Lifecylce Data Sharing using XML, http://www.ugs.com/products/open/plmxml/docs/wp_plm_xml_14.pdf, 2005.
18. Versprille, K. Dassault Systemes' Strategic Initiative: 3D XML for Sharing Product Information, http://www.3ds.com/uploads/tx_user3dsplmxml/3DXML_for_sharing_product_information.pdf, 2005.
19. Vosinakis, S. and Panayiotopoulos, T., A Task Definition Language for Virtual Agents. in *WSCG 2003*, (2003).