An Intelligent Assistant for Computer-Aided Design
Extended Abstract

Olivier St-Cyr, Yves Lespérance, and Wolfgang Stuerzlinger

Department of Computer Science, York University
4700 Keele Street, Toronto, Ontario, Canada M3J 1P3
{olivier, lesperan, wolfgang}@cs.yorku.ca
http://www.cs.yorku.ca/~{olivier, lesperan, wolfgang}

Introduction

This paper describes an approach in improving the usability of computer-aided design (CAD) applications by adding an intelligent agent that assists the user in his/her interaction with the system. To implement the agent, we use ConGolog [GLL97, LLR99] – a very high-level programming language for developing knowledge-based agents that are embedded in complex environments. ConGolog supports the specification of a task-level model of a dynamic environment, the description of complex behaviors, and the synthesis of new plans at run-time. A prototype intelligent agent is being developed to work with an existing 3D CAD system [GS99]. This agent is intended to help the user in designing an office layout that satisfies his goals.

The CAD system [GS99] that our intelligent agent works with is built to allow the user to design a 3D virtual environment (office, kitchen, living room, etc.). The system's graphical user interface is quite simple. Interactions consist primarily of using the mouse to pick and place various types of objects (desk, chair, lamp, inkwell, etc.) in the room layout. In this it resembles the Object Associations system [BS95]. The system handles the details of interactions based on a model of objects and the physical constraints that hold in the scene, for instance, an object being supported by a particular surface. But the system lacks a model of the user, of the task that he is trying to perform, and of the objectives that he is trying to achieve. It cannot really assist the user in quickly creating the desired room layout.

An early example for a system that attempts to aid the user in creating a room layout is CRACK [FM88]. This 2D system critiques the current design with text messages that explain the problem. The domain knowledge embedded in the critiquing system is not used to actively aid the user for placing objects.

We believe that the use of intelligent agent technology can provide many benefits in the area of layout systems. An agent would maintain a high-level representation of the application domain, including object behavior and user knowledge and goals. This could be used to enforce complex application specific constraints on the way objects are manipulated and on the layouts that are produced. Secondly, such a model could be used for disambiguation and consistency checking. Humans often communicate information about a task very inaccurately, because they understand the context of the task and its goals from previous experience. An agent could use its domain knowledge to resolve ambiguities, as well as ask more meaningful questions when user input is required. Moreover, the user goal model could be exploited to detect inadvertent errors. Thirdly, the agent could also aid the user in constructing the virtual design using its knowledge of the domain and user goals. Because it is aware of the current state of the design, it can provide suggestions and advice to the user, guide him through the task, and respond to user questions. All this would lead to much more natural and intuitive interaction between the user and the CAD system.

System overview

Our prototype system consists of three main subsystems that interact and cooperate – see *Figure 1*.

The user interacts with the system through the CAD system's user interface and through a simple graphical user interface dedicated to the intelligent assistant. The main subsystems/components run asynchronously and communicate through TCP/IP sockets. The intelligent assistant component includes the agent's domain model and behaviors specified in ConGolog, and the ConGolog interpreter which runs on top of Quintus Prolog. These tools use logic programming technology, which is well suited to implementing knowledge-based applications. The CAD system component is based on SGI's OpenGL Optimizer 1.1 and implemented in C++. Its primary functions consist in handling constraints during interaction, detecting collisions, positioning objects, etc. These operations are best handled using computer graphics methods as opposed to artificial intelligence methods.

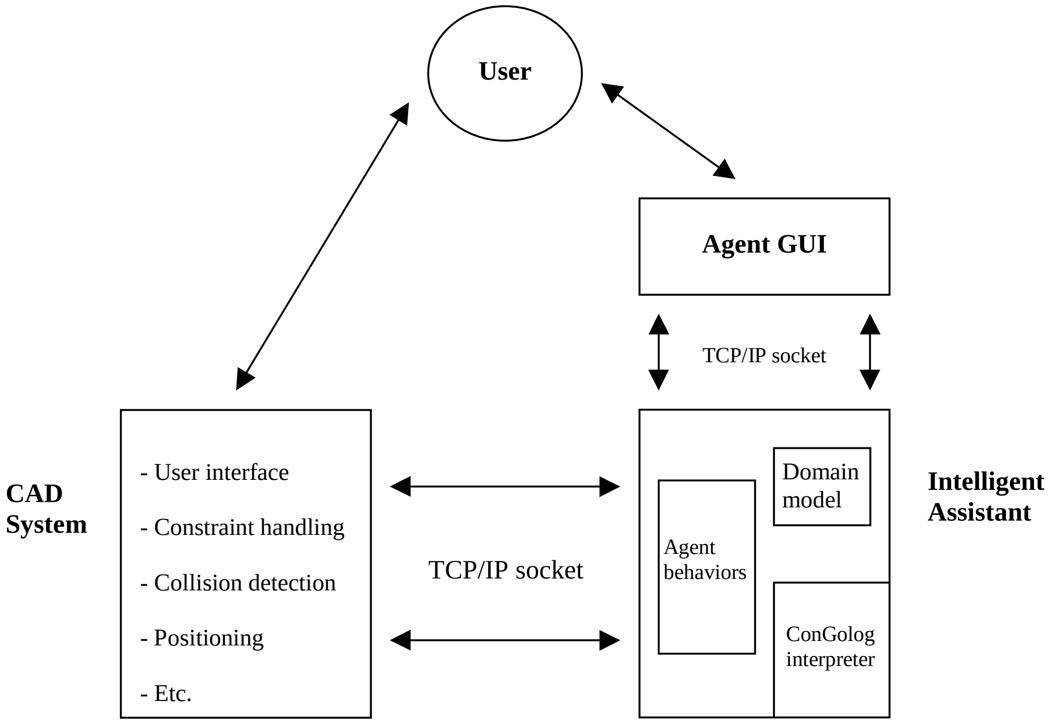

Figure 1. System architecture.

The CAD system

Most of today's CAD systems are well suited to creating geometric objects. Nevertheless, users find common tasks, such as quickly furnishing a room, hard to accomplish, especially since placing objects is conceptually different from creating them. The IConS CAD system [GS99] used in this project is a recently developed 3D application that exploits knowledge about the behavior of objects to provide simple and intuitive interaction techniques for object placement and manipulation. Objects are represented using polygonal models. The application uses a simple two-dimensional interface. The main interaction device is the mouse; keyboard commands are used only for switching modes and organizational tasks. Currently, three modes exist: constrained object movement, unconstrained object movement, and viewer navigation (see [GS99] for details).

A user of this system builds scenes based on a predefined library of objects. For each object, two sets of areas are defined: *offer areas* and *binding areas*. These areas are bound together by constraints and thus, limit the positioning of the constrained object during manipulation. Collision detection/avoidance is also used to ensure that no two objects occupy the same space. These principles (surface constraints and collision detection/avoidance) capture part of the natural behavior of objects in the system. See *Figure 2* for an example of a scene.

General geometric constraints are already part of the IConS system. For example, *OnFloor* and *OnWall* are two of the system's constraints that can limit where an object can be placed. All the implemented constraints describe general placement guidelines for objects that can be used in any design; they are not specific to the application domain. This is where a knowledge-based agent could be useful; it could know for instance, that a computer should not be placed too close to a heat source.

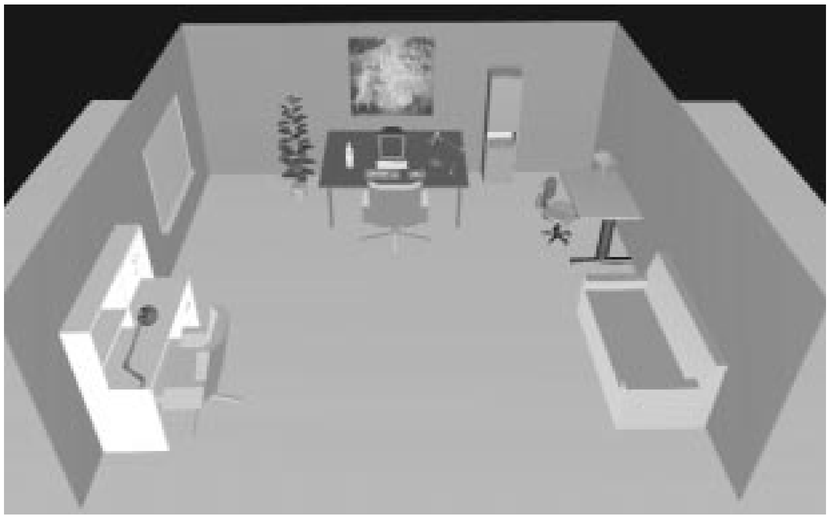

Figure 2. Picture of a scene.

ConGolog

As mentioned earlier, the intelligent assistant agent is implemented in ConGolog. ConGolog is an agent programming language based on techniques for reasoning about action and implemented using logic programming technology. Roughly speaking, one can see it as a concurrent programming language where the primitive actions are not assignments or arithmetic operations, but high-level domain–specific actions like "moving a book onto the desk". A ConGolog program includes two components: a specification of the domain and its dynamics, and a specification of the agent's behavior. The *domain specification* is expressed in a *declarative* formalism based on the situation calculus, a logical language for representing a changing world. It includes declarations for the primitive actions that can be performed by the agent or environment in the domain, declarations for properties of the domain that may change from state to state (these are called fluents), and specifications of the preconditions and effects of the primitive actions in terms of these fluents.

The *behavior specification* is expressed *procedurally* by composing the primitive actions using constructs that include loops, conditionals, concurrency with possibly differing priorities, interrupts, and non-deterministic choice. Behaviors can be complex and the use of domain specific primitive actions means that the specification can be very high-level. A model of the state is automatically maintained by the interpreter based on the domain specification and programs can refer to it by testing conditions expressed in terms of the abstract fluents. The availability of concurrent processes and interrupts facilitates the implementation of agent controllers that monitor and react to changes in their environment while pursuing goal-oriented tasks. Plan synthesis can be performed using non-deterministic search. ConGolog has been used to implement various types of agents including a robot controller [LTJ00] and a meeting-scheduling assistant [LLR99]. Funge [Fu98] has also used a subset of ConGolog to produce more natural computer animations.

The intelligent assistant agent

Domain representation

In the office layout domain, there are many types of objects. These are organized into a hierarchy of classes, e.g. Desk, InkWell, etc. These object classes are themselves instances of a set of function-related metaclasses:

- WorkspaceAreaObjectClass, which includes Desk, Chair, SideTable, etc.
- MeetingAreaObjectClass, which includes MeetingTable, Chair, WhiteBoard, etc.
- StorageAreaObjectClass, which includes Bookshelf, Book, FilingCabinet, etc.
- OfficeEquipmentAreaObjectClass, which includes FaxMachine, Printer, Photocopier, etc.

Spatial relation types (e.g. OnTop, On Floor, OnWorkspace, etc) are also grouped in a hierarchy. There are primitive actions for creating and destroying an instance of an object or relation class.

Another set of fluents and primitive actions is used to represent interactions with the user or the CAD module. For example, the agent can perform the primitive action makeYesNoQuery(msg) and a possible response of the user is represented by the primitive action answerYesNoQueryYes. The declarations for the makeYesNoQuery(msg) are:

```
// preconditions
Action makeYesNoQuery(msg)
possible when ¬ YesNoQueryOut.

// effects
Occurrence makeYesNoQuery(msg)
causes YesNoQueryOut always.
```

These declarations are used by ConGolog to initialize the agent's knowledge base, update it when actions occur, and check the legality of actions in a given state.

Agent behavior specification

So far the behaviors that have been scripted for the agent are rather simple. For e.g., the procedure, which interacts with the user to help him set up the workspace area of his office layout, goes as follows:

```
proc setUpWorkSpace(officeLayout) [
    addObj(Desk,officeLayout);
    addObj(Chair,officeLayout);
    resetYesNoQuery;
    makeYesNoQuery("Would you
    like to have one more chair
    added to your office?");
    yesNoQueryRespIn?;
    if yesNoQueryAnsYes then
    addObj(Chair,officeLayout));
    …
endProc
```

As we extend the agent's behavior, we hope to exploit the distinctive features of ConGolog. For instance, ConGolog makes it easy to use the knowledge base to generate alternatives that satisfy some conditions. As an example, let's assume that the user had just added a desk into his office layout and wanted to add some accessories to it. The agent could produce a menu of alternatives as follows:

```
choose c, o, s , s2, s3 : [
   setOf(c, AccessoriesClass(c), s1)?;
   remove(o, ¬ LegalRelation(OnWorkspace,
      Desk,o), s1, s2)?;
   append(s2, [ "None of the above"], s3)?;
   makeMenuQuery("Please pick an accessory", s3)]
```

Another area where ConGolog helps is in detecting errors and constraint violations. The agent can easily check the legality of the actions requested by the user before performing them. For example, if the user requested the agent to place a computer on top of a heater, it could detect that creating an instance of the OnTop relation between these was illegal. In such situations, it could also suggest alternative actions that may achieve the user's goal. ConGolog's plan synthesis facilities could also be useful for dealing with unanticipated user requests, recovering from failures, or producing animations.

Future work

A simple version of the agent module has been implemented and testing has shown that it can be used to create simple layouts in a straightforward manner. This agent module is now being connected to the IConS CAD system. We are also continuing to develop the ConGolog representation of the domain and extending the behavior of the agent. Our objectives are to demonstrate the feasibility of the approach and document its advantages over standard techniques.

References

[BS95] Bukowski, R., and Sequin, C. 1995. *Object Associations: A Simple and Practical Approach to Virtual 3D Manipulation*, ACM Symposium on Interactive 3D Graphics '95, 131-138, Monterey, CA.

[Fu98] Funge, J. 1998. *Making Them Behave: Cognitive Models for Computer Animation*, Ph.D. Thesis, University of Toronto, Toronto, Canada.

[FM88] Fischer G. and Morch A. 1988. CRACK: A Critiquing Approach to Cooperative Kitchen Design. In Proceedings of the International Conference on Intelligent Tutoring Systems, 176-185, Montreal, Canada.

[GLL97] De Giacomo, G., Lespérance, Y., and Levesque, H. J. 1997. Reasoning about Concurrent Execution, Prioritized Interrupts, and Exogenous Actions in the Situation Calculus. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, 1221-1226. Nagoya, Japan.

[GS99] Goesele, M., and Stuerzlinger, W. 1999. Semantic Constraints for Scene Manipulation. In *Proceedings of the Spring Conference in Computer Graphics*, 140-146. Budmerice, Slovak Republic.

[LLR99] Lespérance, Y., Levesque, H. J., and Reiter, R. 1999. A Situation Calculus Approach to Modeling and Programming Agents. In Wooldridge, M., and Rao, A., editors, *Foundations of Rational Agency*, 275-299. Kluwer.

[LTJ00] Lespérance, Y., Tam, K., and Jenkin, M. 2000. Reactivity in a Logic-Based Robot Programming Framework, to appear in Jennings, N.R. and Lespérance, Y., editors, *Intelligent Agents Volume VI - Proceedings of the 1999 Workshop on Agent Theories, Architectures, and Languages (ATAL-99)*, LNAI, Springer-Verlag, Berlin.