Bowyer: A Planning Tool for Bridging the Gap between Declarative and Procedural Domains

Steven P. Cash and R. Michael Young

Liquid Narrative Research Group
North Carolina State University
Raleigh, NC 27695
spcash@ncsu.edu, young@csc.ncsu.edu

Abstract

Traditionally, there have been two large obstacles faced in attempting to apply AI techniques to games and other virtual environments. The first obstacle is the gap between the largely declarative representations used by many AI techniques and the largely procedural approaches used in virtual environments. The second obstacle is the gap between the skill sets and knowledge bases of the two domain experts with AI researchers often lacking experience using virtual environment APIs and development environments and virtual environments developers often lacking significant AI knowledge. In this paper we present Bowyer, a tool designed to address these two obstacles to the integration of AI planning algorithms into virtual environments. Bowyer bridges the gap between the declarative representations in a planning domain and the procedural framework of a virtual environment via the use of code generation techniques. Bowyer's functionality also allows planning researchers to integrate their planning research into virtual environments without the need to have extensive knowledge of virtual environment development.

Introduction

Traditionally there have been two large obstacles faced in attempting to apply Artificial Intelligence (AI) techniques in games and other virtual environments. The first obstacle is the gap between the largely declarative based domain of AI and the largely procedural based domain of most games, simulations and virtual environments. The second obstacle, that may be largely a reflection of the first, is the lack of virtual environment development knowledge by most AI researchers and conversely the lack of AI knowledge by most virtual environment developers.

Given these obstacles the AI support in most commercial games and other virtual environments have been based on relatively simple procedural AI techniques such as finite state machines (FSM) and variants of the A* search algorithm. Games and other virtual environments can benefit greatly from more advanced declarative AI techniques such as planning. These benefits can be seen in the recent success of the F.E.A.R. (Monolith 2005) commercial video game that uses a planning based AI. As noted by one designer of F.E.A.R. the three main benefits a declarative planning approach gives over traditional procedural AI approaches are: decoupling goals and behaviors, layering simple behaviors to achieve more complex behaviors and adding dynamic problem solving abilities (Orkin 2003).

In this paper we present Bowyer, a tool designed to address these two obstacles to adding AI planning techniques to virtual environments. Bowyer allows a user to specify a declarative planning domain and then automatically generates a procedural representation of that planning domain for use in the virtual environment. Bowyer breaks this process into three general steps. The first step involves the user specification of a planning domain. The second step involves the generation of procedural representations of the planning domain operators and objects for use in the virtual environment through code generation techniques. The final step consists of integrating plans into the virtual environment by a) creating a planning problem, b) automatically retrieving solutions to the planning problem from a planner and c) executing the resulting plans in the virtual environment using the code generated by Bowyer in the second step.

Related Work

In general, Bowyer's functionality can be broken into three sections: planning domain specification, code generation and virtual environment and planner interaction. In this section we present an overview of previous work in these three areas that is most relevant to our discussion of Bowyer.

Planning

The aspects of planning that Bowyer addresses are primarily planning domain specification and planning problem specification. Bowyer's functionality in these areas is based on its integration into the Zocalo planning framework (Vernieri 2006). Zocalo's default planner Crossbow is the C# implementation of the Longbow planner (Young, Pollack and Moore 1994.) which is an extension of the UCPOP planner (Penberthy and Weld 1992) that adds hierarchical planning support. As a result Bowyer's planning domain and planning problem specification support is based on the STRIPS planning language (Fikes and Nilsson 1971).

Code Generation

Code generation is an establish field of research in computer science and has proven to be a useful technique in research based and commercially viable systems. Bowyer's code generation functionality is based on a partial class generation approach. Bowyer uses a code library of small code modules or code "chunks" that can be specified using Bowyer's interface. These code modules are mapped to planning operators and objects and an XML representation is created for the planning operator or object. XSLT templates specific to the virtual environment being used are then applied to the XML to generate the virtual environment source code. Other tools such as Captool (Perrin, Benoit and Foulloy 2002) and the MOmo compiler (Bichler 2003) use a similar approach to code generation to bridge the gap between domain experts and software engineers.

Planning Tools

The planning tools covered in this section are, like Bowyer, tools that were designed to help planning researchers conduct their planning research. We make this distinction to differentiate these planning tools from tools that incorporate planning techniques as an AI technique to accomplish other tasks but are not aimed at improving planning research.

GIPO (Simpson et al. 2001) and its successor GIPO II (Simpson et al. 2003) are designed to aid planning researchers in defining planning domain models. Their functionality includes a graphical interface to define planning domain models, tools for checking the validity of the developed domain models, tools to validate the domain models against existing plans, import and export of planning domains and an interface for integration with external planning algorithms.

ViTAPlan (Vrakas and Vlahavas 2003) is a visual tool used for the highly adjustable planning (HAP) system.
Similar to GIPO, ViTAPlan also supports execution of a plan, checking the consistency of a plan, and similar to Bowyer uses a graphical interface for specifying planning domains and planning problems and generates domain and planning problem specifications.

Bowman (Thomas and Young 2006) is a system that was developed to aid planning researchers, game designers and other users interested in utilizing planning for interactive narrative. Bowman was also built as a part of the Zocalo framework and uses similar planning domain and planning problem specification interfaces as Bowyer.

ScriptEase

ScriptEase (Cutumisu et al. 2007) is designed to be a visual tool for generating scripts for a commercial computer role-playing-game; to create scripted sequences to control non-player characters in the game world. ScriptEase consists of two basic components, the Atom Builder and the Pattern Builder. The Atom Builder is designed to be used by programmers to create small "atoms" of scripting code. The Pattern Builder is designed to be used by designers (nonprogrammers) and uses the atoms to create patterns of actions referred to as situations.

ScriptEase's approach to generation of source code using code modules and common patterns is similar in concept to the approach Bowyer takes to generate code. The use of patterns to generate common source code is prevalent in code generation techniques in general. Bowyer's differentiation from ScriptEase can be seen in its planning support, through its visual specification of planning domains and mapping of planning objects and operators to virtual environment representations for execution of plans in the virtual environment. Bowyer also utilizes an approach to code generation, discussed in the next section, which allows new languages and new virtual environments to be supported without the need to rebuild the Bowyer application code base to support the new virtual environments.

Bowyer Overview

This section provides an overview of Bowyer's functionality. Bowyer's approach to addressing the two obstacles covered in the introduction can be broken down into a series of distinct steps divided into three stages used to translate planning domain representations into virtual environment representations. These steps are represented in Figure 1 grouped by stage.

Also in order to help ground the description of Bowyer, simple examples will be given. The context for these examples is a bank world scenario which includes a bank robber character whose goal is to steal the gold from the bank vault. Due to length constraints this entire scenario cannot be fully described but to aid understanding some examples from this scenario will be given.

Also, while all of Bowyer's functionality is overviewed in this section and available to all users, most users will not use all of the functionality. Planning researchers will generally be able to download the shared resources needed for the virtual environment domain specification and game designers and developers will be able to use and modify existing planning domain and planning problem specifications to gain a better understanding of how to define them.

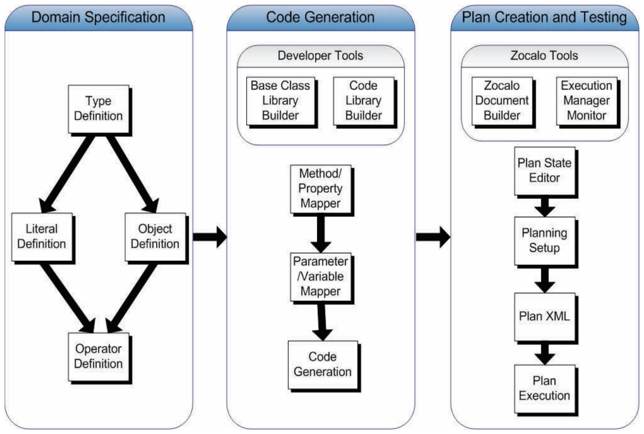

Figure 1: Bowyer Functionality Overview

Planning Domain Specification

Bowyer allows the user to create a planning domain specification using its interface to define the types, objects, literals and operators that exist in the planning domain.

Types in Bowyer represent the types of objects in the planning domain and consist simply of a type name. Types are defined in a hierarchical manner with the root of the hierarchy being the generic type *anyThing* and all types are directly or indirectly derived from this type. One example of a type from the bank world would be the *key* type which is the type of the object that is used to unlock the bank vault.

Objects in Bowyer represent the physical objects and locations in the virtual world that will be used in the planning process. Objects in Bowyer consist of a name, an optional description and can have one or more types assigned to them. The gold in the bank value would be an object in the bank world and would simply need to be an object named "gold" with the type *gold* assigned to it.

Literals are used as the operator's preconditions, constraints and effects and to define the initial and goal states in the planning problem. Literals in Bowyer consist of a name, optional description and a set of arguments for the literal represented by their types. The *has(character, object)* literal is used in the bank world scenario to determine if a character in the world has a given object.

Operators in Bowyer represent the actions that are possible in the planning domain. Operator representations consist of: a name, preconditions, constraints, effects and a set of parameters similar to the way operators are represented in the STRIPS planning language. Preconditions, constraints and effects are represented as lists of literals. A heavily used operator in the bank world scenario is *MoveTo*. This operator has three parameters representing the character that is going to move, the start location and the end location. The preconditions for this operator are *at(character, start)* and *pathExists(start, end)* representing that the character is at the start location and that there is a path between the start and end locations. The constraints for this operator are *isLocation(start)*, *isLocation(end)* and *isCharacter(character)* that verify that the parameters are of the correct type. Finally, the effects of this operator are *at(character, end)* and *notAt(character, start)* which represents that the character has moved and is at the end location and no longer at the start location. These planning domain specifications can be saved and shared so that users not as familiar with planning domain specification can use existing specifications.

Virtual Environment Domain Specification

Bowyer allows the user to specify Zocalo client information for the virtual environment being used. The functionality covered in this subsection is designed for a user familiar with the Zocalo client for the virtual environment, usually the developer of the Zocalo client. The process described here should only need to be completed once for each virtual environment and creates representations saved as XML files that can be distributed with the Zocalo client and loaded by Bowyer. This is one of the steps taken in Bowyer's functionality to help aid nonprogrammer users of Bowyer with the code generation process. The functionality for this subsection includes specifying: the generic operator and object base class representations, code libraries for the virtual environment and the location of the XSLT templates for the virtual environment. This creates a Bowyer virtual environment client specification.

The Bowyer client base classes represent the generic operator and object classes in the Zocalo client, referred to as action classes and world object classes respectively. The definition of the Bowyer client's base classes is necessary for the code generation step to allow the methods in the base classes to be mapped along with code library methods and properties, to the planning operators and objects. The process of specifying base classes consists of specifying the name of each base class as well as the signature for all of the methods in each base class. A method's signature consists of the scope, return type, name and parameters for that method.

Although some functionality for the generated source code classes is inherited from the base classes, additional functionality will be needed in the generated operator and object representations. This additional functionality is obtained from code modules that are saved in code libraries using Bowyer's interface. Each virtual environment must have its own code library built using the programming language that corresponds to that virtual environment. As with the base class definitions the code library is designed to be sharable between users, downloadable with Bowyer and specified by developers for the virtual environment. This portability of the code library is another feature used to aid nonprogrammer users in the code generation process. The code library for each virtual environment is built by adding method and property specifications that can be used to specify possible operator and object functionality. Methods are specified by entering the scope, return type, name, parameters, code body and a description for the method. Properties are specified by entering the type, name and default value of the property.

Finally, Bowyer needs to know the location of the XSLT templates for the virtual environment so that it can use them in the code generation process to create source code representations of the XML representations for the planning operators and objects.

Mapping Between Declarative and Procedural Domains

Bowyer is able to bridge the gap between declarative and procedural domains using code generation and a process of mapping from planning operators and objects attributes to virtual environment code. This includes two steps: mapping code modules to planning operators and objects and mapping code variables to operator parameters. After the mapping is complete the virtual environment code representation of the planning operators and objects can be generated, viewed, edited and saved for use in the virtual environment.

The first step in translating planning operators and objects into their virtual environment representations is to map code methods and properties to operators and objects. For planning operators the mapping consist of mapping the literals for the preconditions, constraints and effects to their method representations as well as mapping the code representations for the operator's execution (the code that actually performs the operator's effects) and any code that should be in the operator's initialization code. For example, a mapping for the *MoveTo* operator would include mapping the *at(character, start)* precondition literal to the *isTouching(actor, location)* method in the code library.

For planning object translation Bowyer allows the user to map any methods or properties to the planning object that reflect the object functionality. For example, the key object would have the *unlock()* method mapped to it.

After the mapping for all of the operators and objects has been completed the next step is the operator parameter to code variable mapping stage.

After all methods have been added to an operator the variables used in each of the method signatures (as method parameters) need to be mapped to the operator's parameters. This is necessary because the variables used in the code represent the world objects in the virtual environment and the operator's parameters are used by the action classes to find the correct virtual environment world objects. A simple one to one mapping is created for each method's variables. For example the *at(character, start)* to *isTouching(actor, location)* mapping given earlier would require a variable to parameter mapping to map *character* to *actor* and *start* to *location* for the generated code to function correctly.

Code generation in Bowyer uses the mapped relationships defined in the previous two steps to create an XML specification of the planning operator or object. This XML specification is then sent to be processed by the current client's XSLT templates to generate the source code to be used in the virtual environment. The generated source code is displayed to the user so that any possible changes can be made such as specifying initial values for properties. The source code can then be saved to file to be used in the virtual environment. This code generation technique was selected to allow Bowyer to generate code for additional virtual environments without the need to modify the Bowyer implementation. This can be achieved by defining a Bowyer client, base class definitions, XSLT templates and a code library for the new virtual environment.

Integration with the Zocalo Planning Framework

The Zocalo planning framework is a web service based framework made up of several independent components. It is designed to facilitate planning support and testing in virtual environments by removing the tight integration between planners and virtual environments. This allows for easy testing of different combinations of planners and virtual environments. The architecture of the Zocalo framework that is relevant to Bowyer's functionality is shown in Figure 2.

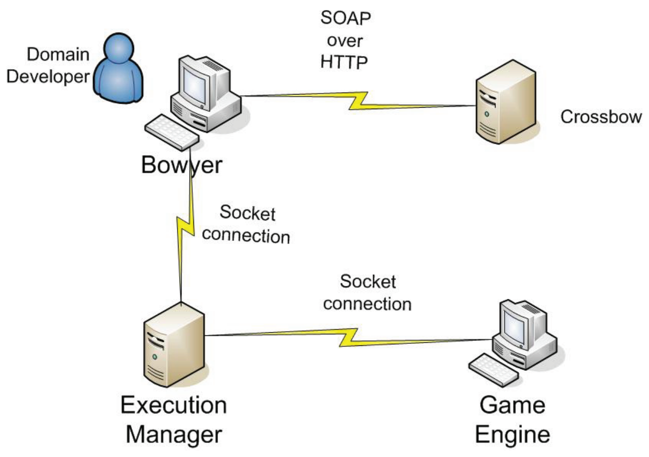

Figure 2: Components of the Zocalo Framework

Bowyer provides functionality to allow the user to graphically specify a planning problem and planning domain, connect to a planner (default is Crossbow), request a plan using the planning problem and planning domain given, and display the returned plan to the user as a plan graph. Bowyer's planner support is based on the planner support that the Zocalo framework provides and therefore requires that planners support the same web service interface that Zocalo uses. This allows planning researchers and game designers to easily swap out planning algorithm implementations.

Before using a planner to find a plan the planning domain and planning problem must be specified for use by the planner. Bowyer allows the user to use the current planning domain maintained by Bowyer or to specify the location of a planning domain document. Bowyer also allows the user to specify the planning problem by specifying the literals in the start and goal state of the plan using the Bowyer interface or by specifying the location of a planning problem document.

Bowyer's planner interaction includes connecting to a planner, specifying the domain and planning problem, specifying some optional plan space search information and retrieving the plan to display to the user. After the plan has been created Bowyer allows the user to connect to the virtual environment through the Execution Manager component and execute the plan in the virtual environment.

Discussion and Evaluation

A small pilot study with five participants was conducted to create an initial evaluation of Bowyer. The pilot study was setup to get potential users, for this study planning researchers, evaluation after giving them an opportunity to interact with Bowyer to complete a set of simple tasks associated with the bank world scenario. The study was broken down into three stages that corresponded to the three sections of Bowyer functionality so that each stage could be evaluated separately. The pilot study used an initial implementation of the Bowyer tool and the Zocalo framework with a client for the Unreal Tournament 2004 game engine.

In general planning domain and planning problem creation is an iterative, trial and error process. In order to conduct the study in a reasonable time period each participant was given a script for the bank world with a set of instructions for steps to complete for each stage. The instructions were written in natural language and included no Bowyer interface specific instructions. The study was meant to give the users a general feel for the functionality of Bowyer and the types of tasks they would be able to complete using Bowyer. At the end of each stage and at the end of the session the participants were given a set of short answer and survey questions to rate their experience using Bowyer and give feedback.

The first stage of the study was used to evaluate Bowyer's ability to allow the user to specify a planning domain. In general the participants found that Bowyer's functionality allowed them to efficiently and intuitively create the planning domain and that it matched well with the STRIPS paradigm. All of the suggested improvements noted by the participants were related to Bowyer's user interface and the fact that defining the types, objects, literals and operators was done separately and is less intuitive then being able to define them together.

The second stage of the study was used to evaluate Bowyer's ability to allow the user to create virtual environment representations of the planning domain operators and objects. The participants stated that while the mapping process was not as intuitive as the planning domain specification process, once the general flow of the mapping process was understood they found it efficient and easily understandable. The main problems found with this stage were that the participants would like more information about the virtual environment code library and a drop and drag interface to do the mapping.

The third stage of the study was used to evaluate Bowyer's ability to allow the user to specify a planning problem, get a plan from the planner and execute the plan in the virtual environment. The participants found the functionality for this stage the most useful in that it would allow them to test planners and plans in the virtual environment and could be used separately with other aspects of their planning research given the ability to save and load planning domains and planning problems. The main concerns and suggested improvements for this stage were usability improvements such as displaying more information about the Zocalo framework and about the current state of the plan and virtual environment.

In general the study participants found Bowyer to be a useful tool that they would use to integrate their planning research into virtual environments, test their research and give demos of their research. Most of the participants' criticisms of Bowyer can be attributed to Bowyer currently being in a proof of concept state with the usability concerns being addressed in future development of the tool. In all the results of the pilot study were promising, showing that Bowyer could be useful to aiding in integration of planning into virtual environments.

Benefits Provided by Bowyer

Bowyer is designed to provide several benefits to the planning researchers and other users integrating planning AI into virtual environments. Bowyer's main benefit is to aid the user in bridging the gap between the planning domain and the virtual environment domain. This provides the additional benefit of allowing planning researchers, and game designers with basic planning knowledge, to incorporate planning techniques and functionality into virtual environments to create more robust experiences, add dynamic narrative structure and allow new elements of narrative and game play to be integrated into the virtual environment that go beyond what is possible with simpler AI techniques. Also Bowyer has been designed to aid planning researchers in testing planning algorithms, plans generated by planning algorithms and other aspects of planning such as interactive narrative in virtual environments by requiring much less knowledge about virtual environment development and allowing researchers to easily swap out components of the testing framework.

Conclusion and Future Work

Future work for Bowyer can be divided into the separate sections of Bowyer functionality.

The approach Bowyer takes to the specification of planning domains can be further improved by adding validation checks for the domain, adding decomposition support for hierarchical planning and adding mediation support to the planning domain specification so that Bowyer can use all functionality provided by the Zocalo framework.

The specification of virtual environment representations as Zocalo client's can be improved by allowing more the one level of inheritance in the generated code as well as allowing the user to use virtual environment classes, previously generated by Bowyer, to be the parent class of newly generated Bowyer classes. The code generation can be improved by adding more intelligence to the code generation process. An additional improvement to code generation is to add functionality to Bowyer to generate planning domain and planning problem specifications in planning languages such as PDDL and STRIPS in the same manner in which virtual environment source code is generated.

The plan generation and execution support can be improved by adding planning mediation support and making the plan execution support more robust to allow for smoother interactions with the virtual environment and more control over the virtual environment.

Bowyer is meant to be a first step towards bridging the gap between declarative and procedural domains. It is designed specifically for planning domains and developed as a proof of concept implementation. Given this, Bowyer is able to accomplish the two goals for which it was designed. It is able to create translations between declarative planning domains and procedural virtual environments and, under the assumption that appropriate virtual environment client specifications and code libraries exist, it is able to allow nonprogrammer users to complete this translation process. Bowyer and its descendent tools should prove to be useful in the development, testing and application of planning research and as a part of the larger Zocalo framework will serve as a robust platform for integrating planning into virtual environments for research based as well as commercial projects.

Acknowledgements

The authors wish to thank the members of the Liquid Narrative research group for their assistance in the creation of the Zocalo framework and for their support in the development of Bowyer.

References

Bichler, L. 2003. A Flexible Code Generator for MOF-based Modeling Languages. *OOPSLA Workshop on Generative Techniques in the context of Model Driven Architecture*. Anaheim, California, USA 2003.

Cutumisu, M.; et al. 2007. ScriptEase: A Generative/Adaptive Programming Paradigm for Game Scripting. *Science of Computer Programming* 67:32–58.

Fikes, R. E. and Nilsson, N. J. 1971. STRIPS: A new Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence* 5(2):189-208.

Monolith Productions, Inc. 2005. *F.E.A.R.,* Los Angeles, California: Vivendi Universal Games.

Orkin, J. 2003. Applying Goal-Oriented Action Planning to Games. *AI Game Programming Wisdom 2*, Hingham Mass.: Charles River Media.

Penberthy, J. S. and Weld, D. 1992. UCPOP: A Sound, Complete, Partial-Order Planner for ADL. *Third International Conference on Knowledge Representation and Reasoning (KR-92),* Cambridge, MA, October 1992.

Perrin, S.; Benoit, E. and Foulloy, L. 2002. Automatic Code Generation based on Generic Description of Intelligent Instrument. *2002 IEEE International Conference on Systems, Man and Cybernetics. Volume: 6.* Hammamet, Tunisia, October 2002

Simpson, R. M.; et. al. 2001. GIPO: An Integrated Graphical Tool to support Knowledge Engineering in AI Planning. *Proceedings of the 6th European Conference on Planning*. 2001.

Simpson, R. M.; et. al. 2003. GIPO II: HTN Planning in a Tool-supported Knowledge *Engineering Environment. Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling (ICAPS 2003)*, Trento, Italy, June 2003.

Thomas, J. and Young, R. M. 2006. Author in the Loop: Using Mixed-Initiative Planning to Improve Interactive Narrative. *In Proceedings of the ICAPS-06 Workshop on AI Planning for Computer Games and Synthetic Characters,* Cumbria, UK.

Vernieri, T. 2006. A Web Services Approach to Generating and Using Plans in Configurable Execution Environments. Masters Thesis. North Carolina State University.

Vrakas D. and Vlahavas, I. 2003. ViTAPlan: A Visual Tool for Adaptive Planning. *Proceedings of the 9th Panhellenic Conference on Informatics*, Thessaloniki, Greece, 2003.

Young, R.M. Pollack, M.E and Moore, J.D. 1994. Decomposition and Causality in Partial-Order Planning. *Proceedings of the 2nd Int'l Conf. AI Planning Systems (AIPS-94)*, AAAI Press, 1994.