A Trivial Study Case of the Application of Ontologies in Electronic Games

#60325

Figure 1: *Ontological Trail*: a game developed through the integration of ontologies and XNA.

Abstract

This paper proposes the creation of a simple architecture to separate programming tasks from resource allocation tasks. To this end, it demonstrates the integration of a knowledge base modeled in DL (Description Logic) with the XNA Game Engine, using the OwlDotNetApi library. The main goal is to demonstrate a trivial use case of ontologies in the domain of games. We have implemented a 2D adventure game using this technique, which we then compare with classical methods of development.

Keywords: ontologies, description logic, XNA

Authors' contact:
xxx

1. Introduction

Techniques derived from ontological knowledge bases have proven efficient for storing information for application in several domains. In the development of electronic games, we may cite dialog creation, textual story generation and AI for adaptive game planning.

This article presents a different application of ontologies: positioning NPCs in an environment and assigning their behavior. We intend to separate the developer's work into programming tasks and level design tasks. During programming, the developer defines procedures related to file loading (graphics, sound, etc.), device control (video, keyboard input, etc.), main game components (collision detection, AI modules, etc.). During level design, the developer defines the NPCs to be inserted in the environment, their habitats (i.e., their positions) and their behavior.

2. Related Work

The main application of ontologies in game design is story flow planning. In [SANCHEZ-RUIZ 2007] the authors present an adaptive approach to game AI through case-based planning and ontological knowledge extracted from the game's environment. There, it is shown that ontology-based extraction results in strategies whose application is easier than those returned by classical mechanisms using only case-based planning.

Like first order logic (using mainly PROLOG), ontologies can also be used for dialog creation in games. In [PEINADO, GERVÁS and DÍAZ-AGUDO 2004] the combination of expedients such as ontologies and formal inference (as featured in Description Logics) has ensured the generation of semantically correct texts. In fact, this mechanism has been explored in automatic story generation influenced by the player's actions. The design of the ontology has allowed measuring the semantic distance between narrative functions for the generation of meaningful stories.

Although current work on games and ontologies demonstrates that this integration can be fruitful, usually little detail is presented about it. Likewise, little information is given about the specification of the knowledge bases and the technologies involved. The present work intends to fill this gap.

3. Ontologies, OWL and DL

Designing an ontology amounts to representing human knowledge in expressive fashion while keeping computational complexity at a minimum.

One of the main ontology languages is the Web Ontology Language (OWL) [W3C 2009]. OWL offers a way to build a vocabulary for the specification of a problem domain, including a set of constructs to define restrictions. Among the main features of OWL are included

- Classes ("things" in the domain of interest);
- Relations (relationships that may occur between things);
- Properties (attributes that things may have).

There are different species of OWL (such as OWL Full, OWL Lite and OWL DL), which vary in complexity and expressivity. The approach presented here is based on OWL DL (Description Logic), which corresponds to a decidable fragment of first order logic [BAADER et. al., 2007]. This species of OWL allows for the processing of complex ontologies with acceptable computational overhead.

4. Tools

The game developed here has been based solely on free software tools. The development environment has been Visual Studio, the base language has been C#, the graphical library has been XNA. Protégé has been used for designing the ontology, and the OwlDotNetApi library has been used to integrate some of those technologies.

We have selected the .NET platform as the main development tool because it supports RAD (Rapid Application Development) and because it allows for interoperability between multiple languages [LIBERTY 2001].

XNA Game Studio Express is an API in the .NET platform that allows easy access to peripherals (such as the keyboard), to the graphics hardware, to audio control and data storage (in files or in databases). This API can also be used as a basis for game development for the XBox 360 console.

For ontology design we have chosen Protégé, which is currently the most popular and powerful tool for this purpose [MULLER 2008]. Protégé can be used to specify domain models and knowledge bases. It allows the creation, manipulation and visualization of knowledge in several different formats, including OWL.

The OwlDotNetApi library [MULLER 2008] is a tool written in C# that offers support to .NET applications. It has the following characteristics:

- It is a C#-based interpreter for the .NET platform;
- It is compatible with the OWL specifications;
- It can be used with any .NET language;
- It can produce directed graphs.

The OwlDotNetApi is available as a precompiled freeware DLL library.

5. Game Description

In order to develop a simple instance of integration of DL and XNA exploring the advantages of ontologies, we have implemented a strategy game. This mechanics is characterized by its emphasis on analysis and reflection in the search for the most appropriate tactics. These are games in which there is an evident component related to territorial or material conquest [SATO and CARDOSO 2008]. We have selected this style because of the diversity in the possible actions by the NPCs (NonPlayer Characters).

The game is called *Ontological Trail* (Figure 1). In it, *Onto*, the main character, must find the three magical relics scattered on *Naufel Island* so he can revive his beloved, who has recently passed away on the island. But he must dodge the wild animals living there (which may or may not attack him). In short, it is a simple game where the player must control a character and avoid or confront certain NPCs while exploring a virtual world.

The software we have developed is highly scalable. Some noticeable characteristics of the NPCs include the following:

- Each NPC has a *type*, an attribute which may be defined in the system. This type may be, for example, *rabbit*, *tiger*, *buzzard*, or *dolphin*.
- Each NPC has a *habitat*, an attribute which may be defined in the system. This habitat may be, for example, *sea*, *coast, jungle, land* or *world* (meaning any habitat).
- Each type of NPC has a random *number of instances,* between 3 and 10, scattered throughout its habitat.
- Each NPC has a random *position* inside its habitat.
- Each type of NPC has an *alignment*, an attribute which may be defined in the system. The alignment may be one of the following: *hostile* (the NPC attacks the main character), *coward* (the NPC runs from the main character), or neutral (the NPC emits sounds upon meeting the main character, but does not attack or run away).

These characteristics are important, as they figure in the links between XNA and the ontology, as described in the next sections.

6. Defining the Ontology

Defining an ontology is not an easy problem [MULLER 2008]. The main steps in the specification of the ontology for the game *Ontological Trail* were:

- Establishing the scope and the main goals of the ontology;
- Defining the conventions for naming classes and properties;
- Enumerating the main concepts and classes;

These principles have been applied to *Ontological Trail* as follows: the scope of the ontology include the animals in the jungle, their habitats and their behavior (alignment). All the classes were named after nouns,

and all the properties had names that were prefixed with the verb form *has* (as in *hasAlignment*). The main classes were organized as shown in Figure 2.

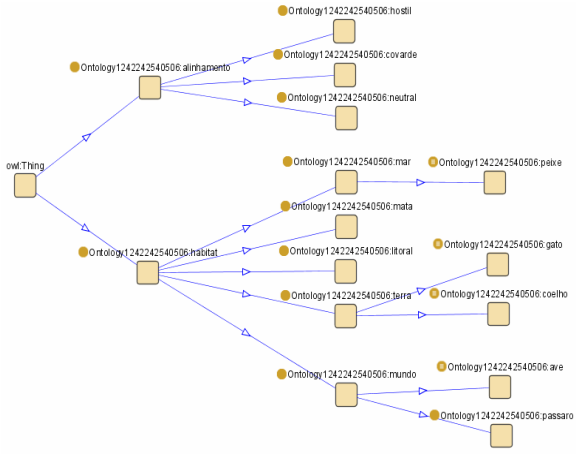

Figure 2: Ontology used in *Ontological Trail*

7. Integration

The following steps have been taken during the implementation of the software:

- Preliminary study of Description Logic;
- Preliminary study of C# and XNA;
- Integration of OwlDotNetApi and C#;
- Definition of the game mechanics;
- Development of the ontology;
- Development of the game using XNA;
- Final integration.

One of the main steps (for its importance and its complexity) was the integration of C# with the OWL knowledge base (Figure 3 shows the first test results). This is explained by the need to use methods *ChildNode( )* and *ParentNode( )* (Figure 5) to extract the relationships defined in DL.

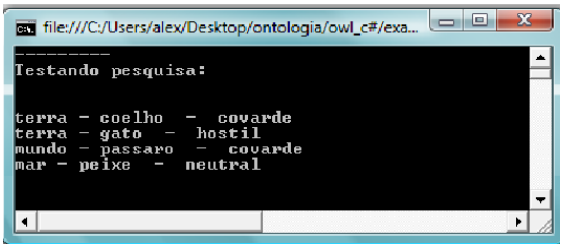

Figure 3: Prototype integrating C# and OWL via the OwlDotNetApi in the Visual Studio console

8. Implementation and Tests

Ontological Trail was developed in Visual Studio and compiled for the PC (Figure 4).

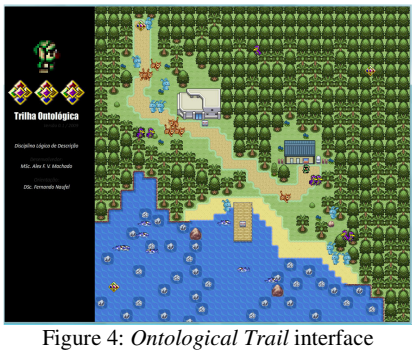

Figure 4: *Ontological Trail* interface

The main classes and components in the developed software were (Figure 5):

- *Animal*: contains the attributes and methods related to the behavior (such as collision detection for the maze walls and for the main character), to the alignment modules (behavior associated to each kind of alignment) and to the rendering of the NPCs. It also handles the external files containing sprites (sequences of images composing a character) and sounds.
- *Habitat.owl*: knowledge base containing the associations necessary for creating an NPC.
- *OwlDotNetApi:* responsible for integrating C# and Owl. Among its main classes we note:
  - *OwlParser:* interprets an OWL knowledge base;
  - *OwlGraph*: generates a connected graph from the knowledge base;
  - *OwlEdgeCollection*: represents a collection of edges. This class maps the identification of each edge into an object of type *OwlEdge* in a list;
  - *OwlEdge:* represents an edge connecting two nodes. It is needed to verify relationships;
  - *OwlNode*: represents a node in the graph.
- *Habitat*: delimits each type of habitat (sea, land, etc.) and enables the generation of the set of NPCs in the game. Uses the *OwlDotNetApi* class to communicate with the *Habitat.owl* component.
- *Game:* main class in the game. Integrates and instantiates all classes in the system.

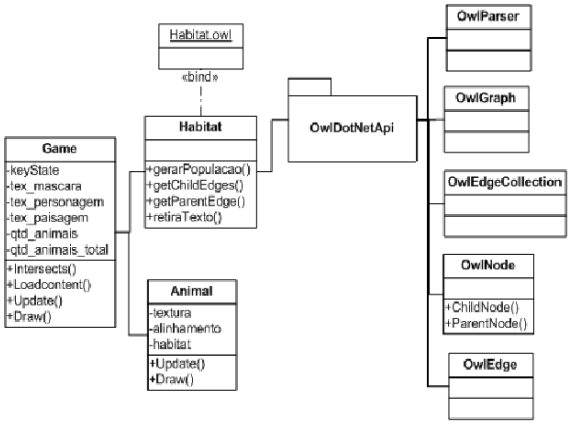

Figure 5: Class diagram

As a consequence of this integration, the developer is free to conduct a substantial part of the level design using Protégé. As shown in Figure 6, it is possible to visualize the NPCs and their habitats in hierarchical form, modify their alignment (changing the restrictions associated to their respective classes) and alter their habitats (by dragging and dropping). To insert a new NPC type, all that is needed is to include its sprite in the project and create a new class in Protégé.

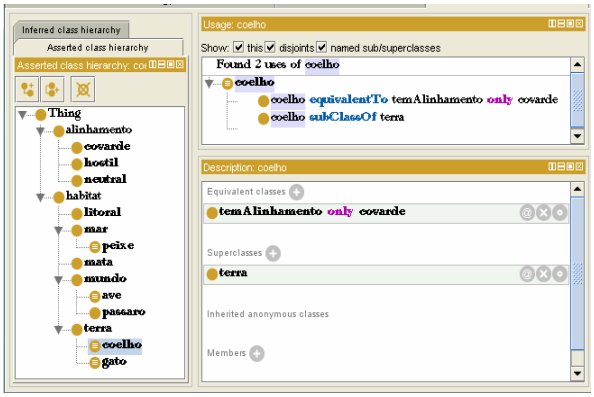

Figure 6: Class hierarchy and equivalences for class *Rabbit* in the game ontology visualized in Protégé

9. Comparison with Classical Models

For this work, we define *classical models* (in the domain of strategy games) as the technique of creating characters in dynamic fashion (using vectors of classes for predefined characters), using hardcoded information (without loading external constants for support).

The difference between this classical model and the ontology-based model is the use of a knowledge base for handling resources (in this case, NPCs).

From these considerations, it is possible to identify the following advantages of our proposed, ontology-based model over the classical models:

- Easier visualization of data and relationships (due to the hierarchical representation);
- Advantages inherited from OOP, such as encapsulation, abstraction and inheritance;
- Easier level design (as the developer is relieved from programming tasks and may focus instead on resource management tasks);
- Easier, more agile maintenance;
- An environment suited to the production of new information related to the resources, as OWL DL reasoners (e.g., Pellet [CLARK & PARSIA 2009]) may be used to infer relationships that were only implicit in the ontology, allowing for semi-automatic refinement of the knowledge base.

10. Conclusion

Although the advantages of the application of ontologies to the domain of games are being widely explored (as in [SANCHEZ-RUIZ 2007] and [PEINADO, GERVÁS and DÍAZ-AGUDO 2004]), little has been detailed about a trivial integration, let alone using free software tools.

The present work demonstrates a step-by-step implementation, from the definition of an ontological knowledge base to the implementation of the main classes of a strategy game. It aims, therefore, at serving as a basic reference for developers interested in starting this integration.

References

BAADER, F., CALVANESE, D., MCGUINNESS, D., NARDI, D., PATEL-SCHNEIDER, P., EDS., 2007. The Description Logic Handbook. 2nd edition. Cambridge University Press.

CLARKE & PARSIA, 2009. http://clarkparsia.com/pellet, last visited July 22, 2009.

LIBERTY, J. Programming C#. O'Reilly, 2001.

MULLER, R., 2008. *Ontologies in Automation.* Vienna University of Technology. Master Thesis.

SANCHEZ-RUIZ A. ET AL, 2007. *Game AI for a Turn-based Strategy Game with Plan Adaptation and Ontology-based retrieval*. Association for the Advancement of Artificial Intelligence.

SATO, A. K. O., CARDOSO, M. V., 2008. *Além do gênero: uma possibilidade para a classificação de jogos*. Proceedings of SBGames'08.

PEINADO, F., GERVÁS, P., DÍAZ-AGUDO, B., 2004. *A Description Logic Ontology for Fairy Tale Generation*. Proceedings of the Workshop on Language Resources for Linguistic Creativity, LREC'04.

W3C, 2009. http://www.w3.org/TR/2009/CR-owl2-conformance-20090611, last visited July 22, 2009.