Game AI for a Turn-based Strategy Game with Plan Adaptation and Ontology-based retrieval

Antonio Sánchez-Ruiz[†] **Stephen Lee-Urban**[⋆] **Héctor Muñoz-Avila**[⋆]
Belén Díaz-Agudo[†] **Pedro González-Calero**[†]

[†]Dep. Ingeniera del Software e Inteligencia Artificial
Universidad Complutense de Madrid, Spain
{antsanch}@fdi.ucm.es, {belend,pedro}@sip.ucm.es

[⋆]Dep. of Computer Science and Engineering
Lehigh University, PA, USA
{sml3, hem4}@lehigh.edu

Abstract

In this paper we present a novel approach for developing adaptive game AI by combining case based planning techniques and ontological knowledge from the game environment. The proposed architecture combines several components: a case-based hierarchical planner (*Repair-SHOP*), a bridge to connect and reason with Ontologies formalized in Description Logics (DLs) based languages (*OntoBridge*), a DLs reasoner (*Pellet*) and a framework to develop Case-Based Reasoning (CBR) systems (jCOLIBRI). In our ongoing work we are applying this approach to a commercial Civilization clone turn-based strategy game (CTP2) where game AI is in charge of planning the strategies for automated players. Our goal is to demonstrate that ontology-based retrieval will result in the retrieval of strategies that are easier to adapt than those plans returned by other classical retrieval mechanisms traditionally used in case-based planning.

Introduction

Developing game AI, i.e. the algorithms that control Non-player Characters (NPCS) in a game, is well-known to be a difficult problem. Three outstanding challenges contribute to this difficulty. First, game developers have little time allocated to develop game AI; other aspects of game development such as storyline, graphics, network connections usually take precedence. Second, the development of environments, called level design, is typically done independently of the development of the game AI. Yet, game AI will be controlling NPCs running in these environments. Third, games change over time. As games are tested, the games are tweaked to improve the gaming experience of the player. This makes constructing effective game AI a moving target.

In this paper we propose a novel approach for developing adaptive game AI. At the core we propose the combination of plan adaptation techniques and ontological information relating objects in the game environment. Such ontological information is readily available in many of these games and is an integral part of their design. This is particularly the case for turn-based strategy (TBS) games. In these kinds of games, two or more opponents (some possibly automated) take turns controlling each's empire or civilization. Controlling these civilizations involves issuing commands to units (e.g., to attack an enemy unit, to defend a location), allocating resources, and constructing new units. To win, various aspects of building a civilization must be taken into account including economy production, assigning resources to improve military and economy, deploying military forces, and using these forces in combat. In TBS games like Civilization or Call To Power hundreds of different kinds of units, buildings, technologies, civilizations, natural resources, terrain features, and weather conditions form an integral part of the game. Game AI must be able to reason with these objects while planning the strategies for the automated players.

Currently, game AI is notoriously famous for cheating to provide an adequate challenge level. So for example, if a city is attacked, the game AI will spawn defenders to meet the challenge even though these defenders are not constructed following the rules of the game. This end up working acceptable in these games as the game AI includes ways to make sure it will not unduly challenge a player. However, cheating does detract from gameplay when the player realizes what is happening. Furthermore, as games are increasingly used as training environments (e.g., for military training or as epistemic games), cheating becomes unacceptable as the simulation environment is flawed.

We propose maintaining a library of known strategies (i.e., plans) in the game. This strategy library includes ontological information about the objects used in the strategies and the conditions under which these strategies where performed. We propose an ontology based retrieval to retrieve *relevant* strategies (e.g., strategies that are applicable to the current gaming situation) and plan adaptation techniques that use the retrieved strategies to adjust them to the current situation. Our goal is to demonstrate that ontology-based retrieval will result in the retrieval of strategies that are easier to adapt than other classical retrieval mechanisms traditionally used in case-based planning.

Next, we describe the most relevant related work and the differences with our approach. The section following that provides an overview of the game and explains why we have chosen it. Then we describe the general architecture of our system and how the different modules are connected. Sub-

sequently, the next two sections provide a detailed description of the main components: the case-based planner and the advantages of an ontological representation of the domain. Finally, we summarize our work and propose future work.

Related work

One application of HTN planning in games is the work of (Hoang, Lee-Urban, & Muñoz-Avila 2005), wherein a team of non-human players in a first-person shooter game is controlled by dynamically generated plans from the ordered-task decomposition planner, JSHOP (Nau *et al.* 1999). This team of bots was shown to outperform teams composed of the same underlying bots that instead used hard-coded strategies. The domain encoding input to the HTN planner, however, was static (pre-encoded); no form of CBR was used. In our work, we intend to retrieve cases of HTN methods that are most suited to the current game state.

Other researchers have applied CBR in games. In (Aha, Molineaux, & Ponsen 2005), cases consist of strategies applicable to specific situations in the real-time strategy game Stratagus. Using these strategies, they were able to beat opponents considered "very hard", such as those that use a "knights-rush". This work differs from our approach in that no form of HTN planning was used nor are retrieved plans adapted.

Another use of case of CBR in games is the multi-layered architecture CARL (Sharma *et al.* 2007). CARL combines CBR and reinforcement learning to achieve a transfer-learning goal in a real time strategy game. Unlike our work, their case retrieval does not use ontological information. Furthermore, case reuse is used to select a goal for the planner to achieve; no plan adaptation occurs.

In (Sánchez-Pelegrín, Gómez-Martín, & Díaz-Agudo 2005) the authors present a preliminary approach that also uses CBR in another clone of Civilization (C-EVO). The AI-module concentrates in a simple tactical problem concerning actions: the selection of a military unit behavior. However, this tactical problem has a big impact on the result of the game. It uses CBR to assign missions to control units based on a big case base of previously assigned missions. Learning and weighting case base features is a challenging problem in this domain because of the difficulties inferring the impact of individual military unit behavior in the final game result.

While all of the above methods employ modern AI techniques in games, to our knowledge, there exists no published work that combines CBP with HTN planning in computer games. This is one of the primary contributions of our efforts.

The planner used in our proposed architecture, Repair-SHOP (Warfield *et al.* 2007), which is capable of performing both plan adaptation and plan repair, is strongly based on other work on plan adaptation, such as (Veloso 1994). Veloso's work takes into account the paths explored by a first-principles, state-space planner, and cases are used to store the failures along with the path that lead to a solution. A similar approach was later implemented for partial order planners (Muñoz-Avila & Weberskirch 1996). Plan adaptation of hierarchical plans has been proposed before ((Kambhampati & Hendler 1992); (Muñoz-Avila *et al.* 2001)). The RETSINA system (Paolucci *et al.* 1999) repairs hierarchical plans in the context of a multi-agent system, but it does not takes into account failed traces. Whereas these works either use solely the path that led to the solution, or failure traces for non-hierarchical planners, Repair-SHOP is distinguished in that it takes into account failure traces for HTN planning.

Some research advocates separating planning from resource allocation (i.e., the information sources). Systems such as RealPLAN (Srivastava, Kambhampati, & Do 2001) and ParcPLAN (El-Kholy & Richards 1996) follow this approach. The separation of planning and resource allocation allows the systems to decompose the problems into parts for which a specialized reasoner is available. We will follow a similar principle of separating a problem into parts and use specialized reasoners for each part – specifically, by separating plan generation from execution.

Game description

Call to Power 2 (CTP2) is a turn-based strategy game (Civilization clone) that was originally developed by Activision and made open-source in 2003. This game is a good choice for our project because gameplay involves many decisions at varying levels of complexity: where to build cities, balancing exploration and expansion with defense, when and where to attack an opponent, which units to produce, which advances to research, etc. The huge amount of possibilities and factors involved in each decision make this environment a great challenge to test techniques like hierarchical planning, case-based planning, and ontological representation of knowledge.

An example is the best way to get an intuitive feel for how CTP2 works. Figure 1 shows the initial state of the game. At this point, the player has two Settler units, which are non-military units that can explore the map and build new cities. A majority of the map, which is a grid of tiles, is unexplored and therefore not visible. When considering a location to build a city, one must consider the type of terrain, amount of food and commerce provided by the map tile, and proximity of tradable goods (e.g. tobacco, beavers) in the vicinity of that tile.

After a city has been built, the decision of number and type of units to produce must be made. In addition to building more settler units, it is possible to produce military units. Alternatively, the city can begin the construction of a *City improvement*, such as a granary. When built, city improvements give benefits to the host city, such as increased commerce or food production, or additional defense against attacks. As the game progresses, another type of improvements, called *Wonders*, becomes available. Wonders (e.g., The Great Wall of China) take many turns to complete and provide special scientific, military or economic bonuses to the player's empire once built.

During the course of the game, enemy civilizations will eventually be encountered, and battles ensue. Combat involves each opposing unit simultaneously dealing damage to each other in turns until one is destroyed or retreats. Bonuses are given to a unit in a defensive stance within a city. For our purposes, game victory is only available through military conquest, which entails destroying all existing enemy units

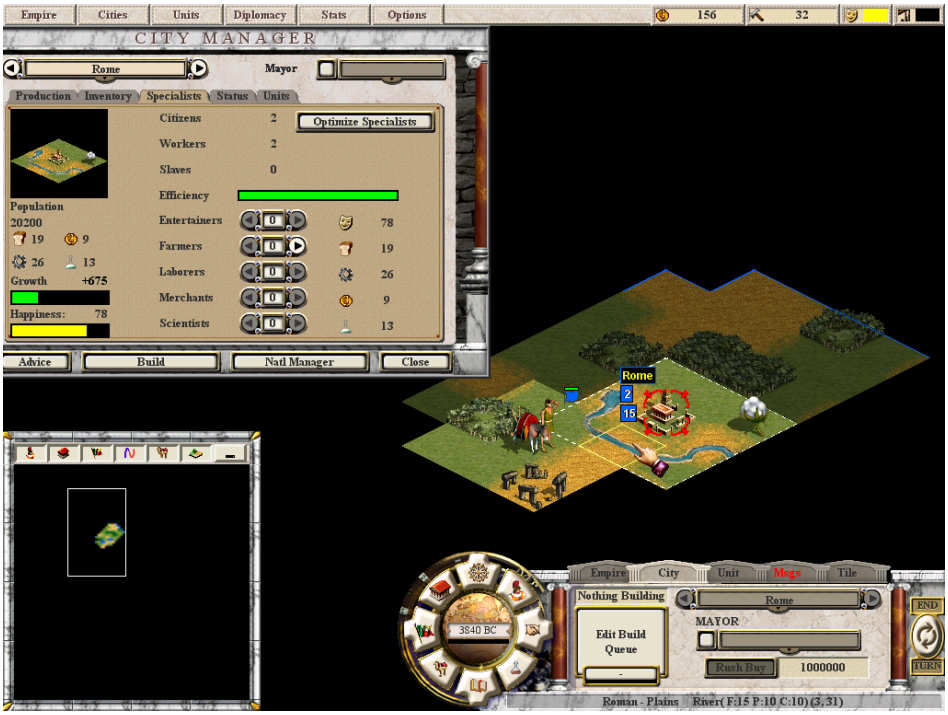

Figure 1: Call to Power 2 (CTP2) is a turn-based strategy game similar to Civilization.

and cities. However, the complete game provides two other means of victory, one through diplomacy through the forming of alliances, and the other through scientific advancement via building a special wonder.

General Architecture

Figure 2 shows the three main modules of our architecture: *Simulator*, *AI Engine* and *Knowledge base*. The communication among these modules is done using well defined interfaces over a TCP/IP connection, so each module can be executed on different hosts.

The *Simulator*, or game engine, is a modified version of the game CTP2, in which a Remote control API has been added to the original game. This Remote control API is described in (Gundevia 2006; Souto 2007) and allows a player to be controlled using a external program. The API is in charge of sending messages to the client about the events that take place in the game (*CityBuilt*, *ArmyMoved*, etc), and at the same time execute the commands that it receives (*BuildImprovement*, *MoveArmyTo*, etc). With this API only a subset of the game functionality is available: city development and improvement, unit production, and military control; however it is enough for our proposes.

The *AI Engine* is the main module of our architecture and it combines several components. Repair-SHOP (Warfield *et al.* 2007) is a case-based planner based on SHOP (Nau *et al.* 1999), a hierarchical planner, and is responsible for generating new strategies. Like any other case-based planner, Repair-SHOP finds new plans by adapting old plans that were successfully used in the past to solve similar situations. jCOLIBRI (Recio-García *et al.* 2005) is the component that finds the most suitable old plan by using ontological knowledge represented in a Description Logic formalism. Actually, jCOLIBRI is a framework for developing CBR Systems capable of managing ontological information using the OntoBridge library (Recio-García *et al.* 2006). Finally, there is a mediator between jCOLIBRI and the ontologies; the DL Reasoner Pellet(Sirin *et al.* 2007), is this mediator, and is responsible for keeping the consistency of the knowledge base and inferring new knowledge that was not explicitly asserted but can be deduced. Once the "best" old plan is retrieved by jCOLIBRI and adapted by Repair-SHOP, it must be executed by the Supervisor. The Supervisor communicates with the game and will both manage plan execution and determine when to begin a new plan retrieval, adaptation, and storage cycle. If the current plan has been accomplished, or is no longer suitable, the supervisor will ask the planner to generate a new plan. In the coming sections we provide a more detailed description of each part of the AI Engine.

The *Knowledge Base* module keeps three different types of information: general knowledge about the domain, used to compute similarity between cases and adapt them, information about the current state of the game, and a collection of old plans indexed by their initial states and goals (cases). The main innovation is that we use ontologies to represent all this knowledge and DLs-like formalism to reason. The main advantages of this approach are discussed in a later section.

As a product of the components of our architecture, the are two main sources of domain knowledge: the tasks, methods and operators used by the planner and stored in the for-

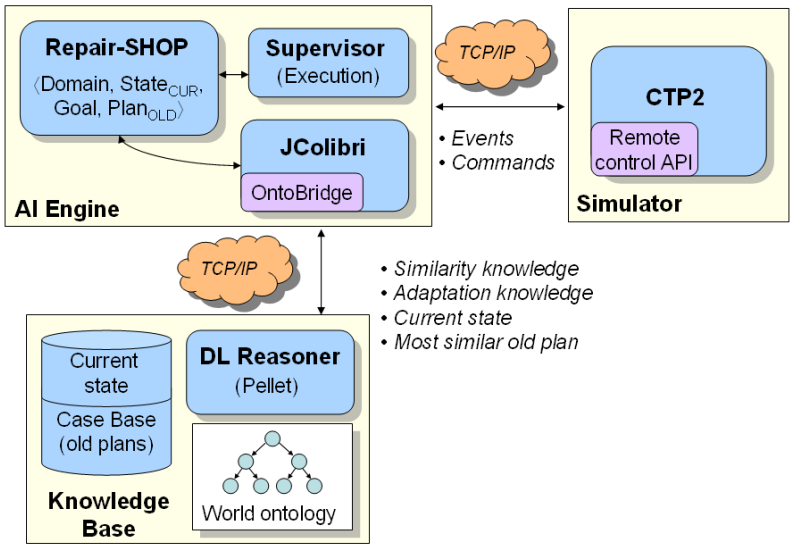

Figure 2: Architecture of our system and main modules

malism of Repair-SHOP; and the knowledge about the different types of objects in the world and their relations that are stored using an ontology. In addition, the current state must be cloned in both formalisms because Repair-SHOP and jCOLIBRI use it in different ways: the planner must keep the current state to decide what methods and operators are applicable to adapt an old plan; and jCOLIBRI keeps the current state in an ontology to retrieve the most similar case using similarity metrics based on hierarchies and DLs.

Planner

In plan repair an existing plan must be modified because of changes in the world conditions (van der Krogt & de Weerdt 2005). Repair-SHOP, a planning system built on top of the HTN planner SHOP, is capable of performing plan adaptation and plan repair (Warfield *et al.* 2007). Among its distinguishing characteristics is the ability to take into account failed traces, which can result in improvements in running time performance. The HTN planner SHOP implements a variant of HTN planning called Ordered Task Decomposition. In this variant tasks are totally ordered and conditions are evaluated relative to the current state of the world, which is updated during planning.

A case used in Repair-SHOP is defined as the tuple (T,S,GG): a collection of HTN tasks T, a state S, and a graph structure GG, called the goal graph. The goal graph GG represents the HTN generated when solving (T,S), augmented by other relations, and takes the form of a directed dependency graph with a one-to-one mapping between each goal in the graph and each task in SHOP. This graph represents relations between goals, operators and decisions (applied operators). The GG, which is an implementation of the REDUX architecture (Petrie 1991) with HTN planning in mind, maintains dependencies among SHOP task nodes, allowing SHOP to monitor changes in a task's preconditions. This structure propagates changes in conditions to the appropriate task nodes; thus, SHOP can replan the affected sections through dependency-directed backtracking.

Upon input of a case (T,S,GG) and a new problem (T',S'), Repair-SHOP uses the case's GG relative to (T',S') in order to generate an HTN for the new problem. The same domain model is implicit in both the new problem and case. Dependencies are evaluated relative to (T',S') resulting in a partial HTN that is completed by using standard HTN planning techniques.

The advantage of using the GG alongside SHOP is that GGs preserve information about the state of the plan for each task and subtask that SHOP attempts to solve. Leaves in the GG correspond to primitive tasks in the HTN. Internal nodes in the GG correspond to compound tasks in SHOP, culminating in the original compound task at the root of the GG.

Repair-SHOP's operation is straightforward. When Repair-SHOP monitors a change in conditions, it propagates the result to the highest affected goal, and then checks for an alternate decision. If no alternate decision is available, the graph is navigated upwards towards the root until the first alternate decision is found. If an alternate decision is eventually found, the stored SHOP state from that decision is restored and the SHOP planning algorithm restarted. If no plan can be found, Repair-SHOP searches upward in the GG for a new alternate decision. If finally successful, the new plan is saved and then spliced into the original plan beginning with the first affected goal node.

There are still areas in which Repair-SHOP could be improved. Specifically, the system can only consider situations wherein conditions in the case are invalid. Clearly, it would be desirable to consider situations where new conditions are added (e.g., additional resources are made available).

Knowledge Base and Similarity

As we have already explained, CTP2 is a very complex game in which the player must manage several different types of resources: units, city improvements, technical advances, wonders, etc. Furthermore, each time a player uses his turn the game time will advance, so a normal game will cross different ages (stone age, ancient age, modern age, etc). In each age different features are available. Clearly, the complexity of this environment is substantial: there exist hundreds of features that interact in different ways depending upon the stage of gameplay.

All this information can be intuitively described using taxonomies. Actually, the game user documentation includes several tables and graphs in which all these resources are classified. The use of these tables and graphs represent a suitable way to describe complex worlds and so this kind of documentation can be found in several strategy games. We chose to use ontologies, as a generalization of taxonomies, to represent not only the subclass relations but a more complete description of the domain. Ontologies are an expressive and standard mechanism to represent reusable knowledge that has been successfully used in several areas.

It is evident that these ontologies keep knowledge, and our goal is to use this information in the retrieval and adaptation phases of our case-based planner to improve the performance and accuracy. This way, the planner is able to use two main knowledge sources: the cases, that represent concrete past experiences, and ontologies, that represent general knowledge about the domain. In this sense, our approach can be described as a knowledge-intensive case-based planner (KI-CBP).

In figure 3 a small part of our ontology is presented (the current ontology uses more than 60 defined concepts and 400 instances to represent the world). Each entity of our world can be classified using different criteria. For instance, units are classified by the environment in which they operate (ground, water, air), the age they become available (ancient, renaissance, modern, etc) and by their military features. In the same way, the advances are classified by the age and the technological area (Construction, Economics, Cultural, Medicine, etc). We will use this ontology to compute similarities between different entities. Intuitively, two entities will be more similar the closer they are in the hierarchy and the more parent concepts they share.

Let us remember the whole retrieve process of our case-based planner. Assume that Repair-SHOP needs to build a new plan to achieve some goals in the current state of the world. At this point, Repair-SHOP asks the jCOLIBRI component for the most similar case in the case base to achieve those goals in the current state. All the previous plans are stored in an ontology indexed by goals and initial states. Then, using the classification capabilities of DLs, a set of previous cases is retrieved using only some primary features. After that, the retrieved cases will be ordered using more accurate numerical similarity functions and the most similar case is returned to the planner for adaptation. These similarity metrics are discussed in next section.

The process we describe is quite complex compared to the standard foot-printing similarity metric used in CBP, which only counts the number of equivalent predicates in the states. However, we think that by using complex retrieval metrics we will get better cases that will be easier to adapt, reducing the adaptation time and improving the quality of the final plan. We can measure the accuracy of the retrieval as the inverse of the plan adaptation effort.

DL and Ontology-Based Similarity

Description Logics (Baader *et al.* 2003) are a set formal languages (subsets of First Order Logic) that are typically used to formalize ontologies and reason with complex worlds. This formalism has been studied for several years and its features are well defined. An ontology is compound of a *TBox* (terminological information or concepts) and a *ABox* (asserted information or instances). The TBox contains the concepts, roles (relations between concepts) and their definitions, and the ABox contains the instances. For example, the concept *Ancient_Advance* is defined as the intersection of the concepts *Advance* and *Ancient_Item*. In the same way, an *Ancient_Item* is defined as anything with a role *hasAge* with the value *AG_Ancient*.

$$Ancient\_Advance \equiv Advances \sqcap Ancient\_Item$$
$$Advances \equiv \{AD\_Agriculture\ AD\_Chemistry\ \ldots\}$$
$$Ancient\_Item \equiv \exists hasAge.\{AG\_Ancient\}$$
$$Ages \equiv \{AG\_Ancient\ AG\_Renaissance\ \ldots\}$$
$$\ldots$$

We use the TBox to represent the domain constraints, i.e. the domain information that will not change, and the ABox to represent the current state of the world. This way the concept *City* will represent the terminological definition of what is a city in our domain, and its instances will represent the current cities in an specific state of the game. The idea of using an ontology to represent the state in planning has been previously proposed in (Sirin 2006), for the Semantic Web Services domain. However, in that work they use a different approach than ours for the planning part.

The main features of DLs are that they can automatically check the consistency of the ontology (if there exists at least one model for the ontology), and they can classify new concepts and instances. The consistency checking is useful when creating the ontology to check that there are no "impossible definitions". The reader must take into account that we are modeling very complex worlds and it is very easy to make mistakes. The automatic classification of new concepts and instances classifies the information about the new states and will make easier to compute similarities (González-Calero, Díaz-Agudo, & Gómez-Albarrán 1999; Salotti & Ventos 1998).

The main two approaches to compute the similarity using ontologies are:

- *Classification based retrieval using DL classification capabilities*. A new concept is built with the common properties that we are looking for, then this concept is classified and its instances are retrieved. Another variation is to start with an instance, look for the most specific concepts of which this individual is an instance, and then retrieve all the instances of those concepts.

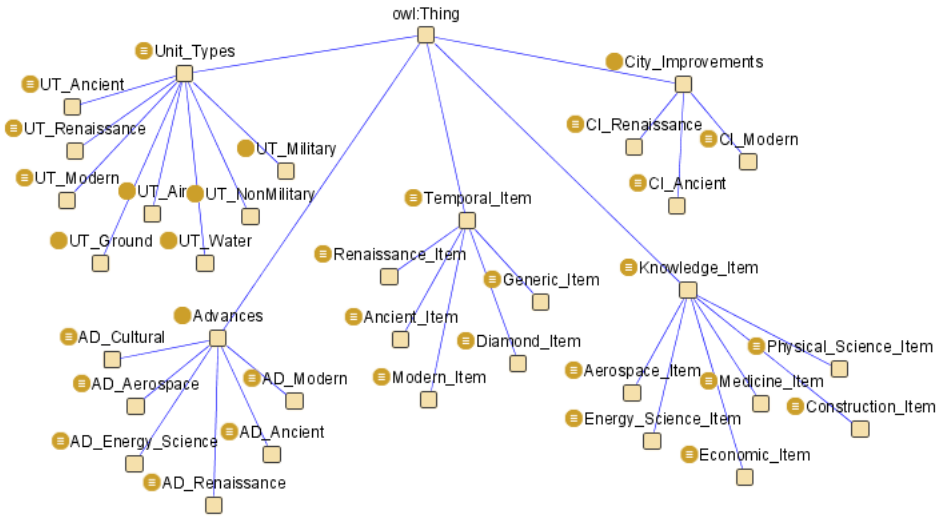

Figure 3: Small part of the ontology used to represent the domain in the game CTP2

- *Computational based retrieval*. In this approach numerical similarity functions are used to assess and order the instances. The use of structured representations of the knowledge requires similarity metrics to compare two differently structured objects, in particular, objects belonging to different classes. Usually the similarity is determined recursively in a bottom up fashion (Bergmann & Stahl 1998), i.e., for each simple attribute, a local similarity measure determines the similarity between the two attribute values, and for each complex attribute a recursive call compares the two related sub-objects. Then the similarity values returned are aggregated (e.g., by a weighted sum) to obtain the global similarity between the initial objects. Different weights can be used to represent the varied importance of properties in the similarity measure.

In general, the similarity computation between two structured cases can be divided into two components that are aggregated (Bergmann & Stahl 1998): the computation of a concept based similarity that depends on the location of the objects in the ontology (or *intra-class similarity*) and the computation of a slot-based similarity (or *inter-class similarity*) that depends on the fillers of the common attributes between the compared objects. Figure 4 shows the intra-class similarities implemented in jCOLIBRI .

In our system we will use both approaches. The classification based retrieval will retrieve a set of most similar cases to our current problem taken into account only the most important features, and then, numerical similarity functions will be used to order those retrieved cases using more accurate metrics. All this functionality will be provided by the framework jCOLIBRI that has already implemented all these similarity metrics.

Here we propose a very trivial example in which the advantages of our approach in terms of quality are easy to check. Imagine that in the current state of the world we have 2 units: a warrior and a submarine, and our goal is to destroy an enemy city. In our case base there are only two cases with the same goal. In case 1 there is only one unit, a warrior. In case 2, there are two units: a knight and a destroyer. If the foot-print similarity is used for retrieval, then case 1 will be selected because there is 1 match (the warrior unit) against 0 matches with the second case. However, case 2 is in actuality more similar because a knight is similar to a warrior (both are from the ancient age and ground military units) and a submarine is similar to a destroyer (modern age and water military units). Using the ontological approach all of these similar qualities will be taken into account and case 2 will consequentially be selected.

Current Status and Future Work

In this paper we have presented our ongoing effort to integrate case-based planning techniques with knowledge intensive case-based reasoning using ontologies. In order to reduce the size of the search space, most approaches to planning represent the state of the world using propositional logic. In complex domains, such as real time strategy games, the expert may find more natural ways to describe the world using object-oriented representations with inheritance.

One of the main features of Repair-SHOP is its ability to accelerate the planning process. It does so by reusing previous plans that can be efficiently adapted when some conditions required by the plan are not met in the current state. The ultimate goal of the work presented here is to determine whether using a rich representation of the domain for retrieval purposes may result in gains on speed and accuracy of the planning process.

Speed ups may come from a more accurate retrieval that results in reduced adaptation effort and therefore reduce the time to obtain a plan, or from the reduction of the size of the case base resulting from a more expressive language to describe the cases. The accuracy of the case retrieval process will need to be measured in terms of the adaptation effort required along with the performance (i.e., game score) obtained by the resulting plans.

$$\text{fdeep\_basic}(i_1, i_2) = \frac{\max(\text{prof}(\text{LCS}(i_1, i_2)))}{\max\limits_{C_i \in CN}(\text{prof}(C_i))} \qquad \text{fdeep}(i_1, i_2) = \frac{\max(\text{prof}(\text{LCS}(i_1, i_2)))}{\max(\text{prof}(i_1), \text{prof}(i_2))}$$

$$\text{cosine}(i_1, i_2) = \text{sim}(\text{t}(i_1), \text{t}(i_2)) = \frac{\left| \left(\bigcup_{d_i \in \text{t}(i_1)} (\text{super}(d_i, CN)) \right) \bigcap \left(\bigcup_{d_i \in \text{t}(i_2)} (\text{super}(d_i, CN)) \right) \right|}{\sqrt{\left| \bigcup_{d_i \in \text{t}(i_1)} (\text{super}(d_i, CN)) \right|} \cdot \sqrt{\left| \bigcup_{d_i \in \text{t}(i_2)} (\text{super}(d_i, CN)) \right|}}$$

$$\text{detail}(i_1, i_2) = \text{detail}(\text{t}(i_1), \text{t}(i_2)) = 1 - \frac{1}{2 \cdot \left| \left(\bigcup_{d_i \in \text{t}(i_1)} (\text{super}(d_i, CN)) \right) \bigcap \left(\bigcup_{d_i \in \text{t}(i_2)} (\text{super}(d_i, CN)) \right) \right|}$$

CN	is the set of all the concepts in the current knowledge base
super(c, C)	is the subset of concepts in C which are direct or indirect superconcepts of c
LCS(i_1, i_2)	is the set of the least common subsumer concepts of the two given individuals, i.e., the most specific concepts both individuals are instances of
prof(c)	is the depth of concept c, i.e., the number of links in the longest path from c to the top concept following the concept-superconcept relation
t(i)	is the set of concepts in CN the individual i is instance of
prof(i)	= max(prof(t(i))), where i is an individual

Figure 4: Concept based similarity functions in jCOLIBRI

At this point we have successfully integrated in a single architecture the game CTP2 with Repair-SHOP and jCOLIBRI , demonstrating the technological feasibility of the approach. From this point, we are going to start building a case base large enough to run the experiments that would demonstrate the benefits of the approach. Cases may be obtained by recording actual users playing the game and by using Repair-SHOP to generate plans by running it against human or artificial oponents.

References

Aha, D. W.; Molineaux, M.; and Ponsen, M. J. V. 2005. Learning to win: Case-based plan selection in a real-time strategy game. In Muñoz-Avila, H., and Ricci, F., eds., *ICCBR*, volume 3620 of *Lecture Notes in Computer Science*, 5–20. Springer.

Baader, F.; Calvanese, D.; McGuinness, D. L.; Nardi, D.; and Patel-Schneider, P. F. 2003. *The description logic handbook: theory, implementation, and applications*. New York, NY, USA: Cambridge University Press.

Bergmann, R., and Stahl, A. 1998. Similarity measures for object-oriented case representations. In *EWCBR*, 25–36.

El-Kholy, A., and Richards, B. 1996. Temporal and resource reasoning in planning: The parcPLAN approach. In Wahlster, W., ed., *Proc. of the 12th European Conference on Artificial Intelligence (ECAI-96)*, 614–618. Wiley & Sons.

González-Calero, P. A.; Díaz-Agudo, B.; and Gómez-Albarrán, M. 1999. Applying DLs for retrieval in case-based reasoning. In Lambrix, P.; Borgida, A.; Lenzerini, M.; Möller, R.; and Patel-Schneider, P. F., eds., *Description Logics*, volume 22 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Gundevia, U. 2006. Integrating war game simulations with ai testbeds: Integrating call to power 2 with tielt. Master's thesis, Lehigh University.

Hoang, H.; Lee-Urban, S.; and Muñoz-Avila, H. 2005. Hierarchical plan representations for encoding strategic game ai. In Young, R. M., and Laird, J. E., eds., *AIIDE*, 63–68. AAAI Press.

Kambhampati, S., and Hendler, J. A. 1992. A validation-structure-based theory of plan modification and reuse. *Artificial Intelligence* 55(2-3):193–258.

Muñoz-Avila, H., and Weberskirch, F. 1996. Planning for manufacturing workpieces by storing, indexing and replaying planning decisions. In *AIPS*, 150–157.

Muñoz-Avila, H.; Aha, D. W.; Nau, D. S.; Weber, R.; Breslow, L.; and Yaman, F. 2001. Sin: Integrating case-based reasoning with task decomposition. In *IJCAI*, 999–1004.

Nau, D.; Cao, Y.; Lotem, A.; and Muñoz-Avila, H. 1999. Shop: Simple hierarchical ordered planner. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, 968–973.

Paolucci, M.; Shehory, O.; Sycara, K. P.; Kalp, D.; and Pannu, A. 1999. A planning component for RETSINA agents. In *Agent Theories, Architectures, and Languages*, 147–161.

Petrie, C. 1991. *Planning and Replanning with Reason Maintenance*. Ph.D. Dissertation, University of Texas at Austin, Computer Science Dept.

Recio-García, J. A.; Sánchez-Ruiz, A. A.; Díaz-Agudo, B.; and González-Calero, P. A. 2005. jCOLIBRI 1.0 in a nutshell. a software tool for designing CBR systems. In Petridis, M., ed., *Proccedings of the 10th UK Workshop on Case Based Reasoning*, 20–28. CMS Press, University of Greenwich.

Recio-García, J. A.; Díaz-Agudo, B.; González-Calero, P. A.; and Sánchez-Ruiz, A. A. 2006. Ontology based CBR with jCOLIBRI. In Ellis, R.; Allen, T.; and Tuson, A., eds., *Applications and Innovations in Intelligent Systems XIV*, 149–162. Springer-Verlag London.

Salotti, S., and Ventos, V. 1998. Study and Formalization of a Case-Based Reasoning system using a Description Logic. In *EWCBR*, 286–297.

Sánchez-Pelegrín, R.; Gómez-Martín, M. A.; and Díaz-Agudo, B. 2005. A CBR module for a strategy videogame. In Aha, D. W., and Wilson, D., eds., *1st Workshop on Computer Gaming and Simulation Environments, at 6th International Conference on Case-Based Reasoning (ICCBR)*, 217–226.

Sharma, M.; Holmes, M.; Santamaria, J.; ; Irani, A.; Isbell, C.; and Ram, A. 2007. Transfer learning in real-time strategy games using hybrid cbr/rl. In *Proc. of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*.

Sirin, E.; Parsia, B.; Grau, B. C.; Kalyanpur, A.; and Katz, Y. 2007. Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics* 5(2).

Sirin, E. 2006. *Combining Description Logic reasoning with ai planning for composition of web services*. Ph.D. Dissertation, University of Maryland.

Souto, J. 2007. A turn-based strategy game testbed for artificial intelligence. Master's thesis, Lehigh University.

Srivastava, B.; Kambhampati, S.; and Do, M. B. 2001. Planning the project management way: Efficient planning by effective integration of causal and resource reasoning in realplan. *Artificial Intelligence* 131(1-2):73–134.

van der Krogt, R., and de Weerdt, M. 2005. Plan repair as an extension of planning. In *Proceedings of the International Conference on Planning and Scheduling (ICAPS-05)*, 161–170.

Veloso, M. M. 1994. *Planning and Learning by Analogical Reasoning*, volume 886 of *Lecture Notes in Computer Science*. Springer.

Warfield, I.; Hogg, C.; Lee-Urban, S.; and Munoz-Avila, H. 2007. Adaptation of hierarchical task network plans. In *Proceedings of the Twentieth International FLAIRS Conference (FLAIRS-07)*.