A Semantic Layer for Knowledge-Based Game Design in Edutainment Applications

Andrea Repetto, Chiara Eva Catalano
CNR-IMATI Ge
Via De Marini, 6
16149 Genova, Italy
Email: {andrea.repetto,chiara.catalano}@ge.imati.cnr.it

***Abstract*—Creating and maintaining complex and realistic virtual worlds is still a challenge in game design. Realism is not only related to visual appearance but also to the interactions and situations in the game. This issue is particularly crucial in edutainment applications where realism impacts the learning aspect of the game experience. Introducing semantics in virtual worlds helps define intelligent objects and interactions which would turn into a more realistic game. In this work, we propose to decouple the semantic definition of the game world from its actual implementation in a general-purpose game engine. A semantic layer has been developed to bridge the semantics formalized by ontologies with its realization in the engine. Thanks to this software library, semantics can be specified in a separate formal module and reused in different projects. The proposed approach has been tested to design a serious game concept set in the marine environment.**

Keywords*—*Semantics, object interaction, ontologies, edutainment, natural heritage.

I. Introduction

Recent advances in game development technologies have made it possible the creation of virtual worlds with a high level of realism, both in visual appearance and in interactions. In games for leisure, realism helps immersion and engagement of the player; in edutainment applications, it may be not only desirable but even crucial to represent real-world situations and convey some specific message to the player. Realism in the game world is usually due to the skills of developers, able to express it in the environment, in the characters and in the interactions, and encode it explicitly in the game. Introducing the concept of semantics in games helps to obtain such realism more "naturally" [1].

According to [2], semantics in the context of virtual environments is defined as "the information conveying the meaning (of an object) in a virtual world". Semantics can be defined at different levels: the *object semantics*, which is related to the single object; the *object relationship*, which describes the possible interactions between objects; the *world semantics*, which is about the status of the whole world. These three levels are interconnected.

In a typical game development scenario, ad-hoc solutions are used to define the semantics of a virtual object, such as mark-up languages or behavior scripts. Thus, object semantics and object relationships are hard-coded, and this represents a limitation in terms of achieving "intelligent", i.e. natural and automatic, interactions in a virtual world, and ultimately in the development of emergent gameplay mechanics.

A widely adopted technology to represent semantics is given by ontologies and can be also applied to express formally the semantics of virtual objects in a game context. Gruber [3] defined an ontology as "a formal specification of a shared conceptualization". Ontologies provide a flexible, yet machine-understandable, form to express a hierarchy of concepts, called classes, and how they relate to each other by a set of logical axioms. An ontology may also contain a set of individuals, which are instances of some classes. Individuals constitute the ground level of ontologies: they are concrete objects, for which ontologies provide a formalization. The combination of an ontology and the corresponding set of individuals forms a knowledge base. Besides, using a software component called semantic reasoner, additional knowledge encoded by axioms can be exploited.

From a technical standpoint, a game is developed by means of a game engine, which contains all the functionalities related to the various aspects of the game, like input, graphics, sound, assets, and behavior. A game company usually either develops its own engine or uses a general-purpose one. In both cases, the adoption of ontologies to express the semantics of objects and relationships in a virtual world poses the problem of translating such representation into the engine to design intelligent objects and interactions.

In this paper, we present the semantic layer we developed to create a bridge between the semantic representation of the game elements and their usage (i.e. semantic data) in a general-purpose game engine. Thanks to this software library, we are able to:

1) **Specify semantics independently of a particular game engine**: this is made it possible by the semantic layer, which associates the semantic representation of game objects to the corresponding objects in the game engine;
2) **Provide semantic data to the game engine in a machine-understandable way**: this allows high-level reasoning on the role of an object inside the game world, making its interaction capabilities with other objects potentially higher;
3) **Modularize semantic data**: instead of structuring the knowledge base as a monolithic entity, we could keep the aspects related to different knowledge domains sep-

INTETAIN 2015, June 10-12, Torino, Italy
Copyright © 2015 ICST
DOI 10.4108/icst.intetain.2015.259561

arate. This allows to compose them according to the developer's needs, and replace only the module related to a single aspect of the game, while leaving the others in place;

4) **Allow the reuse of the formalized semantics for different software projects**: this allows the game designer to use or extend a set of predefined properties and behaviors of the game objects, instead of rewriting them from scratch for each development project.

These features are advantageous when creating complex game environments in order to decouple semantics and gameplay, and are especially beneficial in edutainment and serious gaming applications. In such games "used for non-leisure purposes" [4], the role of semantics is central. On the one side, it involves the realism both of the virtual world and of the situations with respect to the educational purpose. The latter is often much more significant than the first: the main goal in this context is conveying a message and it is this one that should guide the design of the game and of the virtual world. On the other side, the learning content has to be transmitted correctly and appropriately to the target audience: the semantics to associate to characters and to all the game objects has to adhere to the instructional objective.

Formalizing such content in an ontology corresponds to the creation of a separate *educational* module, which constitute the base for a pedagogically-driven design of the game. Moreover, a learning module as such is reusable for different applications or different games. Indeed, reusability is currently an open issue in the development of serious games, because these are often developed with a low budget, while involving stakeholders with different expertise, that is institutions/schools/companies on the one hand, and game developers on the other hand. Reusable learning modules equipped with a suitable semantic layer to bridge the gap with game engines appears a promising approach to face such bottleneck.

The paper is organized as follows. In section II, we describe the previous works in the definition of semantic objects and interactions in a game, organized from the lowest to the highest level of semantics they aim to express. In section III, we give a description of the semantic layer we propose. In section IV, we describe the game concept for an edutainment application we are developing and using to validate the approach. In section V, we discuss the technical aspects of the implementation. Finally, in section VI we show the advantages and limits of our work, and give an overview of possible future research directions.

II. Previous works

In the introduction, we have stressed the importance of designing an effective serious game in order to convey a specific message to the player. Achieving this goal is not an easy task for a game developer, who does not usually have competences in the learning domain knowledge, in pedagogy and psychology and would need some guidelines to set up the project correctly. In our research, we focused particularly on methods employing a formal specification of the learning content in order to drive the design process; thus, the problem is to translate such knowledge in the game world appropriately.

There are two phases where a semantic description can be beneficial: the *design* phase and the *execution* phase. In the design phase, some constraints on the placement of the game objects can be imposed semantically [5]; alternatively, the world can be generated automatically with knowledge-based procedural techniques [6] [7].

Our work, however, concentrates on the executive phase of the game, where semantics can be employed for the specification of the interactions. Indeed, the game industry has already recognised the importance of a semantic representation in game AI, suggesting, as a best practice, the usage of an intermediate layer of knowledge between the agent and the game world in order to achieve a more life-like behavior [8].

In academia, a first step in this direction goes back to the definition of smart objects [9]: smart objects are virtual objects enriched with embedded functionality; according to the definition given in [2], they belong to the object semantics level. Intelligent agents can interrogate smart objects to plan a series of actions [10], collaborate with each other [11], or to identify possible interactions on the base of previous experiences [12].

The STARFISH architecture [13] defines synoptic objects for real-time object manipulation by autonomous agents. A synoptic object can provide agents with "a summary, or synopsis, of what interactions it affords". Although the agent is able to do some reasoning on the object properties, inspired by the concept of affordance [14], the communication language is specified ad-hoc for the specific application.

Smart objects have influenced many commercial games: the *Sims*[1] game series is a notable example. It is a life simulation game regarding a family of virtual humans that express an autonomous behavior. Objects around them can be used to satisfy their needs (e.g. hunger and sleep): this is possible because each object offers information on which need can satisfy and how it can be used (in concrete terms, which animation to play). *Scribblenauts*[2] represents another interesting case study. The main character has the ability to create objects by selecting from a dictionary of nouns and adjectives, where each term is mapped to a hierarchy of classes and the adjectives qualify dimensions and functions; each item is provided with its typical behavior/functionalities. For instance, a ladder can be created bigger/smaller and has the function to be climbed.

Another recent trend in reaching a new level of interaction appeared in *sandbox* games. *Minecraft* is an example, where worlds are generated procedurally. These are composed of cubic "blocks" that the user can collect with the appropriate tools. The blocks can be reused later, either to craft other tools or to build structures. Each element of the game world is in fact an interactive game object.

[1]http://www.thesims.com
[2]http://www.scribblenauts.com

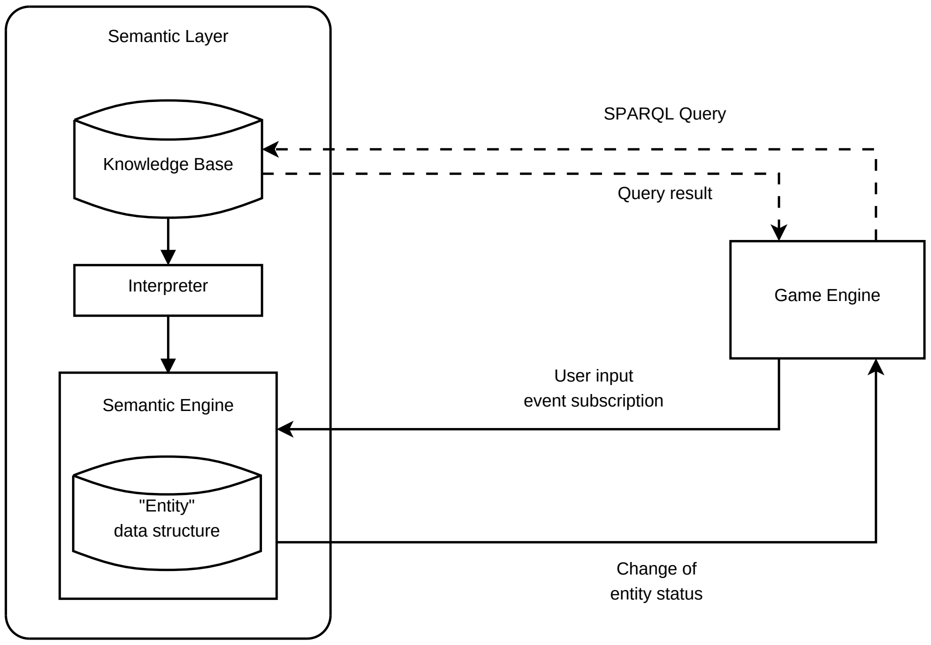


Fig. 1. The Semantic Layer architecture.

All the cited examples, despite their commercial success, display a limitation: virtual objects do not capture the actual nature of the real object they represent, and the associated semantics is limited to the system for which it has been developed.

Even though specifying ad-hoc interactions between virtual objects is acceptable for many applications, we are interested in exploring more general approaches, which use a standard formalism for describing interactions.

In [15], the authors used an ontology as a formal language to describe the interactions between virtual objects. A specific component, called *causal engine*, intercepts the events occurring in the physics engine, and replaces them with unexpected ones, in order to follow the artistic purposes of the game. Although the formal specification (or part of it) can be reused in different projects, the type of interactions that the system can intercept are limited to the physics engine scope.

In [16], the authors proposed a more general approach, where interactions between virtual objects are specified in terms of the services they provide to each other, according to a class hierarchy defined by an ontology. A software component called *semantic engine*, takes care of the execution of these interactions at run-time. The authors populated a large repository of virtual objects from Wordnet [17], a lexical database of nouns, adjectives, verbs and adverbs. The terms have been used to populate the several categories of the repository: nouns have become entities, verbs have become actions, and so on. However, this mapping has been defined statically; if a developer wanted to include in the game a knowledge domain with different characteristics, an ad-hoc mapping would be required.

In our work, we aim to be even more general, defining a mapping mechanism that translates an arbitrary domain ontology to the service paradigm: instances of an arbitrary ontology are translated to game entities, relying on the service paradigm for the definition of smart interactions. A software library, that we call *semantic layer*, acts as a “wrapper” between the ontological knowledge base and a general-purpose game engine.

III. Semantic Layer

A. Overview

Thanks to the semantic layer, the game engine is able to:

- Access the semantic data defined in the knowledge base;
- Interrogate game objects about their role and establish interactions based on their semantics;
- Take actions when the status of an object changes in the semantic layer, such as changing its visual appearance or behavior.

Since the ontology in this context includes a formal specification of the game objects and how they interact with each other, a subset of the individuals has a direct correspondence with the entities in the game. For instance, we may have an ontology where the class `Person` defines specific characteristics for a human being, which has to be reflected in the human characters of the game world. In practice, we define a correspondence between the instances of the class `Person` (along with its attributes) and the game entities that represent persons. Fixing such correspondence requires at least three steps:

1) Retrieving the instances of the class `Person` in the knowledge base;

2) For each instance, retrieving the property values of interest (name, height, relationships with other persons and so on);
3) Instantiate game entities according to these values.

Writing an ad-hoc procedure for such a "conversion" from individuals to game entities is unfeasible: in fact, as the number of properties and classes grows, the process becomes more and more complex and time consuming. Our contribution is the automation of this process.

Such automation is possible if we put in relationship the abstract behavior defined in the domain ontology with its concrete realization inside the game. Since we do not want to fix constraints on a specific technology or game engine, we followed a more general approach for the concrete realization of interactions. We found that the the service paradigm was a suitable choice [16], [18], [19]: indeed, one of its characteristics is that interactions are not handled directly by the game engine: a black box component, called *semantic engine*, is responsible for this.

The process that translates the abstract behavior in the domain ontology to its concrete realization inside the game is described as it follows (see fig. 1).

First, the game designer defines a mapping between the domain ontology and the service paradigm, formally specified with an ontology as well.

Before the game is run, the domain ontology together with the mapping to the service paradigm are merged in a single ontology; moreover, a semantic reasoner derives other axioms that are implicitly encoded in the original scheme. The result is saved in a single OWL file. Further details of the mapping mechanism are given in section III-B.

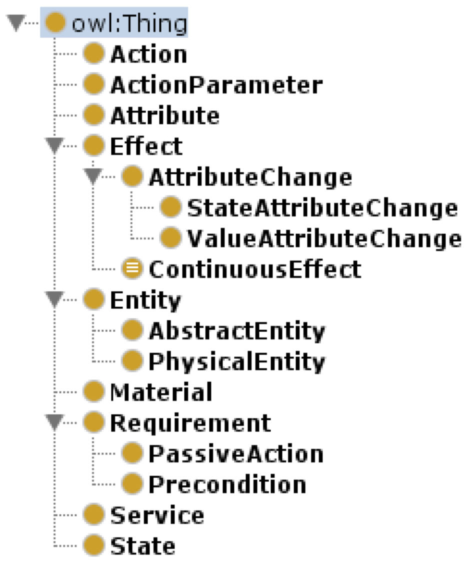
At the beginning of the game execution, an initialization phase takes place:

1) An interpreter reads the OWL file and initializes the semantic engine. The semantic engine is supported by an in-memory data structure: it is basically a "working copy" of the instances of the knowledge base, used for efficient computation (see section III-C);
2) The game engine links each entity in the semantic engine to a corresponding game object in the game engine (its "physical" representation); with an event system, the game engine is able to detect when an entity in the semantic engine changes its status, and respond accordingly (e.g. playing an animation).

Then, in the main loop of the game engine, an update of the semantic engine is requested at fixed time intervals, which may cause changes to the internal state of the support data structure. When this happens, the game engine is notified, according to the event handlers defined previously. Events happening in the game engine (e.g. user input, collisions) are passed to the semantic engine, so that the corresponding entities in the data structure are updated accordingly.

We point out that, in some case, the game developer may be interested to accessing the data contained in the knowledge base directly: although this is not the primary purpose of the library, methods for running SPARQL queries (of SELECT type) on the knowledge base are provided for full flexibility.

The following sub-sections detail the mapping mechanism, which translates the abstract behavior to the service paradigm, the interpreter and the semantic engine.



```
owl:Thing
    Action
    ActionParameter
    Attribute
    Effect
        AttributeChange
            StateAttributeChange
            ValueAttributeChange
        ContinuousEffect
    Entity
        AbstractEntity
        PhysicalEntity
    Material
    Requirement
        PassiveAction
        Precondition
    Service
    State
```

Fig. 2. The class hierarchy of the services ontology.

B. Mapping an arbitrary ontology to the service paradigm

The mapping mechanism, which happens in a preprocessing phase, is based on translating concepts of an arbitrary ontology to those of the service paradigm: for this reason, we formalized the service paradigm in an opportune ontology (fig. 2).

Each ENTITY has a number of ATTRIBUTES that may change over time, and offers a number of SERVICES to other ENTITIES; SERVICES are activated under certain REQUIREMENTS, and produce some ACTIONS in response. ACTIONS produce certain EFFECTS on the ENTITIES, typically a change to an ATTRIBUTE value.

In this ontology, we have an `Entity` class. Each instance of `Entity` is treated as a game entity; the `Entity` class has two subclasses: `PhysicalEntity` and `AbstractEntity`. Each object having a physical representation inside the game is an instance of `PhysicalEntity`, thus occupies a position in the 3D space and is subject to physics laws. Instances of `AbstractEntity`, in contrast, do not have a physical representation: typical examples are a `Team` in a football game, or a `Nation` in a strategy game.

The mapping ontology defines which classes of the knowledge domain should be entity classes in the service paradigm. This is done by using the subclass relationship.

Since the concept game we are developing is set in the marine environment, we will describe a typical example: a fish needs to search for food when hungry. In the domain ontology, the `Fish` class is defined; each instance of `Fish` (and its subclasses) is declared also as a game entity by adding the axiom

`Fish is_a Entity`.

Datatype and object properties require a more sophisticated mapping, because typically they do not have a one-to-one correspondence with the concepts of the service paradigm.

A rule language is able to derive the appropriate mapping between the two. Among the possible choices, we have used the SWRL [20] language: some semantic reasoners are able to process these rules without requiring additional software packages.

The `Fish` class, in the domain ontology, is described by the following properties:

- A `maxNutritionLevel` property, which expresses the "level of nutrition " of a satiated fish (as a numerical value);
- `hungerThreshold` property: if the nutrition of the fish is below this threshold, the fish is considered hungry;
- Finally, animals may have one or more behaviors, expressed by the property `hasBehavior`. All `Fish` instances have a `LooksForFoodWhenHungry` value, for this property: as implied by the name, it means that when the fish is hungry, it looks for food.

The game designer can define appropriate rules that map these properties to services.

First, we create an attribute called `NUTRITION` that encodes the current nutrition level of the fish (which changes over time). The initial value of this attribute will be the same of `maxNutritionLevel`:

```
maxNutritionLevel(?x,?v) ->
  hasAttribute(?x, NUTRITION),
  NUTRITION(?x, ?v)
```

An analogous rule will map the `hungerThreshold` property to a corresponding `HUNGER_THRESHOLD` attribute.

The abstract behavior `LookForFoodWhenHungry` will be mapped to appropriate services:

```
hasBehavior(?x, LookForFoodWhenHungry) ->
  providesService(
  ?x, Service_LookForFoodWhenHungry)
```

The mapping mechanism that we have described results more general and reusable if compared with an ad-hoc solution (e.g. behavior scripts): the same mapping can be adapted to a different domain ontology, or the same domain ontology may have different mappings in different games.

C. Interpreter

At the end of the pre-processing phase, which includes the mapping mechanism we described in the previous sub-section, an OWL file is produced: this file becomes an asset of the game.

The semantic engine relies upon a data structure, composed of **`IEntity`** objects (see fig.3, bottom). Each one of them is a "working copy" of an instance of a `Entity` class in the ontology; also, **`IEntity`** objects are responsible of the event-based mechanism between the game engine and the semantic engine, typically each time a service is activated/deactivated, or an attribute changes its value.

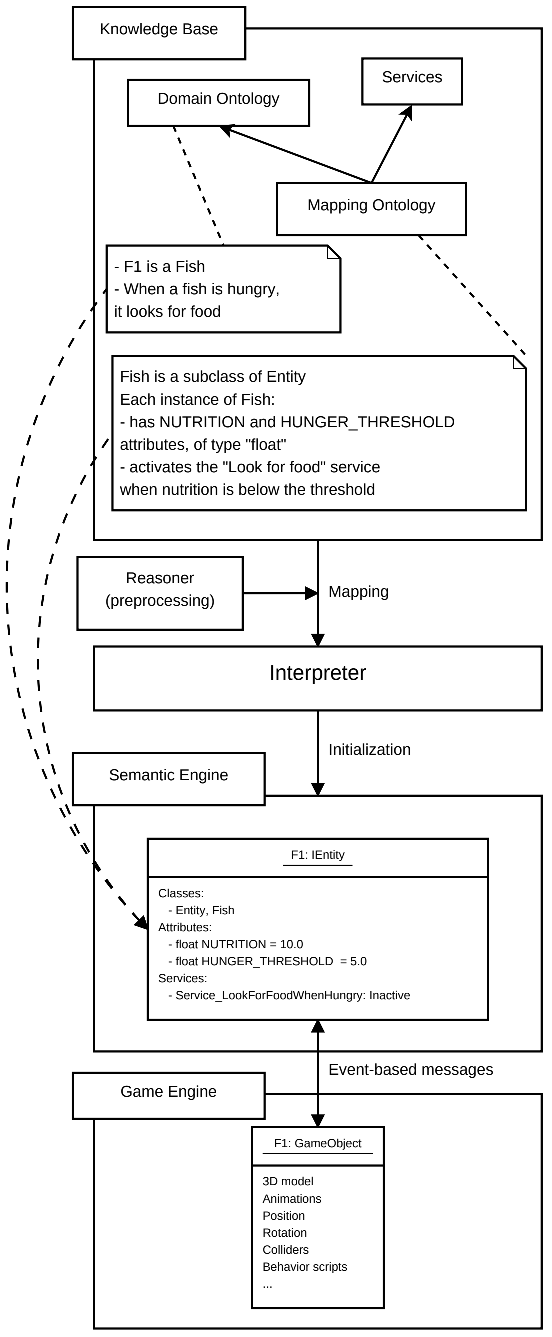


Fig. 3. An instance of `Fish` in the domain ontology is translated to its run-time counterpart as a game object.

During the initialization phase of the game, for each instance of a subclass of the `Entity` class in the knowledge base, a corresponding **`IEntity`** object is instantiated, along with its attributes and services.

Having an **`IEntity`** object guarantees the actual independence of the game engine, where a corresponding game object will be instantiated. The two objects are linked together, and in this way it is possible to establish a message exchange between the game engine and the semantic engine, when specific events occur (i.e. a collision event between two semantic game objects triggers a `Collision` action in the semantic engine).

In the interpreter, for each entity class, we retrieve the attributes defined on that class through the `hasAttribute` property. For each attribute associated to the `hasAttribute` property, we retrieve the type (which can be a primitive value or a state) and we create a corresponding attribute object (**`IAttr`** interface).

Default values of attributes are not specified in classes, to avoid possible conflicting default values on the same attribute because of the multiple inheritance. Oppositely, the solution proposed in [5] was to let the game designer take care of the conflict by manually overriding the default value. Instead, we specify the initialization values as property assertions on individuals. This choice also gives designers a higher degree of flexibility: if they want to instantiate an entity with different default values, all they have to do is to define an alternative individual, with different property values.

D. Semantic Engine

At this point, the interpreter has built a data structure which contains the game entities, and their interactions expressed by services.

As we have already mentioned, services are a way of expressing rules about the run-time behavior of game entities; such rules are basically `if-then` constructs, where the `if` part is made of requirements, and the `then` part is made of actions. At each update step, the semantic engine evaluates these rules, and performs the corresponding changes to the entities when needed.

These rules are quite different from the logical axioms encoded in the ontology, because the OWL language is a subset of first-order logic, which has a higher expressive power than if-then rules; on the other hand, OWL reasoning has a higher computational cost, unfit for real time computation. A semantic reasoner is however used to infer knowledge contained in the domain ontology, as part of the mapping phase that we described earlier. The use of the inference engine is thus limited to a pre-processing phase.

We use actions as a form of "message exchange" between entities, in the sense that an action cannot modify directly an attribute of another entity. Only the receiving entity can do it, as it "knows" how to handle a certain type of action. In other words, a certain action may have different consequences, according to the objects that receives the action.

To give an example of how interactions can be modeled as an exchange of "action" messages, we consider a fish which increases its nutrition level each time it eats some food. The sequence of steps that are performed in the semantic engine are:

1) `Fish`, subclass of `Animal`, performs an `Eat` action on `Food`;
2) `Food` receives the `Eat` action;
3) In response, `Food` sends a `IncreaseNutrition` action, with a `NutritionValue` parameter (this happens only for instances of `Edible` class);
4) `Fish` receives the `IncreaseNutrition` action and in response increases its nutrition level of the amount specified from the `NutritionValue` parameter.

This example shows that we have not defined any hard-coded interaction between the `Fish` and the `Food` entities: however, the interaction is established automatically. It has to be noted that the semantic engine does not handle completely the interaction in this case: although the described sequence of actions happens automatically, a behavior script in the game engine triggers the action `Eat` of the `Fish`. This example shows also the advantage of reusing an existing class taxonomy: the behavior of the class `Animal`, the `Eat` action here, is inherited by all the instances of all the subclasses of `Animal`, as we would expect in a real-life situation.

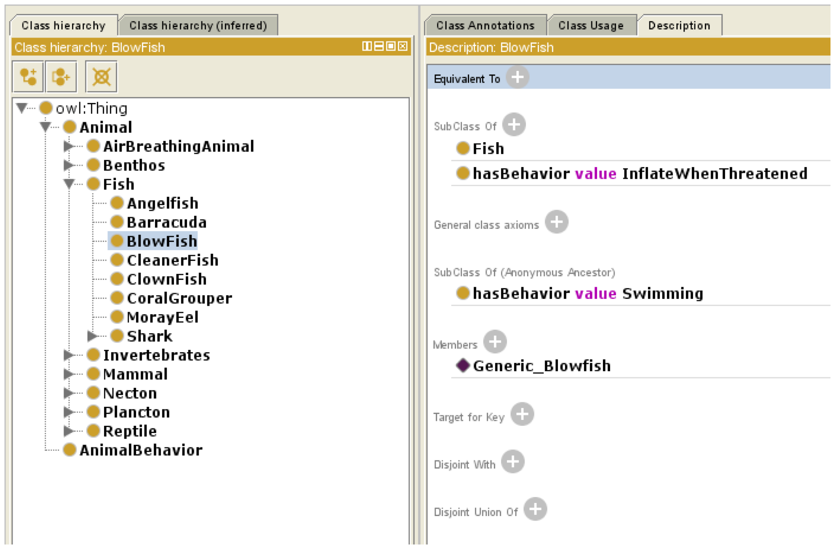
A final remark is that the semantic engine replaces partially the functionalities of traditional behavior scripts. Some are still needed but their usage in the presented system can rely on higher level information on the object, and therefore handle interactions in a more natural and intuitive fashion.

IV. Game concept: a serious game set in the marine environment

We tested the effectiveness of our approach in the development of a concept for a serious game targeting 8-12 year old children to raise awareness about the marine environment and its safeguard.

In the project, the game should contain knowledge of the marine environment, in particular the identification of the marine species and their characteristics with respect to its habitat. To this purpose, we decided to define a light ontology from scratch including the species taxonomy together with their features, since the learning content should be simple enough to be suitable to the target players. This taxonomy and the properties associated can be used in conjunction with the behavior expressed through the service paradigm. In a bigger project, we might think of reusing existing sources of biological taxonomies shared in the scientific communities, since our library is able to read any ontology written in the OWL language.

In this concept, we have imagined two possible game scenarios. In the first scenario, the game presents to the player a certain marine environment with a short textual description: it may be either the coral reef or the open sea. The child's task is to choose from a catalog the marine species that are suitable to live in that environment. The color, size and shape of a marine organism gives some hints on its living habitat: this phenomenon is known as *convergent evolution*: species with a different lineage, but living in the same environment, in their evolution have developed similar characteristics. The kid has to look at the features of the different species to select the right one for the given habitat. The catalog of species is actually the knowledge base, where the instances are the game objects corresponding to the various species.



Class hierarchy: BlowFish

```
owl:Thing
  Animal
    AirBreathingAnimal
    Benthos
    Fish
      Angelfish
      Barracuda
      BlowFish
      CleanerFish
      ClownFish
      CoralGrouper
      MorayEel
      Shark
    Invertebrates
    Mammal
    Necton
    Plancton
    Reptile
  AnimalBehavior
```

Description: BlowFish

Equivalent To

SubClass Of
- Fish
- hasBehavior value InflateWhenThreatened

General class axioms

SubClass Of (Anonymous Ancestor)
- hasBehavior value Swimming

Members
- Generic_Blowfish

Target for Key

Disjoint With

Disjoint Union Of

Fig. 4. A screenshot of the Protégé editor that shows the marine-species ontology. It defines the class taxonomy of the marine species, and their possible behaviors.

The second scenario addresses the behavior of marine species and how they relate to each other. The player is given some quests, which involve taking "virtual pictures" of the marine species, while they display some peculiar behavior. Some behaviors could be related to the species itself: for instance, all mammals need occasionally to come back to the surface to breathe air; some others are related to interactions between species, such as the symbiotic relationship between the clownfish and the sea anemone. The marine species ontology defined includes also such information.

The interactions are implemented with the support of the semantic engine. The game engine is still in charge of initiating the interactions. In our marine environment, interactions between species may happen when one entity "perceives" another, or in practical terms, when they are close enough.

The advantages in terms of design are the following:

- The ontology of the marine species is machine-understandable and is used for computation inside the game;
- The semantic engine encapsulates part of the behavior of marine animals, leaving the game engine the focus on presentation and user interaction;
- The marine species ontology is reusable in different software projects, because the semantic layer has been designed to be independent of the adopted game engine.

In the project, we have defined three different ontologies, which can be seen as modules. Each module describes a different aspect of the marine environment; however, these aspects are in relationship with each other:

- *marine-species*: it contains the description of the marine species as a biological hierarchy and their behavior. `Fish` classes include properties like color, shape and size. In the formalization of the behaviors, these are classified according to how they move in the water: *nekton* are swimming animals, *plankton* are carried by currents, and *benthos* live on the sea bottom. Besides, each species may have its particular behavior: for example, the blowfish inflates when it is threatened by a predator (see fig. 4).
- *marine-habitats*: it contains a description of the habitats (in our project, open sea and coral reef); there are also the axioms which relate habitats with their inhabitants.
- *marine-services*: it is a mapping ontology, which translates the abstract behaviors in terms of services. This is necessary to model situations of states that change dynamically during the execution of the game. In the example of the blowfish, the state from `Inflated` to `Deflated` changes according to the presence of a threat. This ontology references both the *marine-behavior* ontology and the *services* ontology.

These ontologies have been defined with the support of a marine biologist and pedagogist, who have provided us with the learning content about sealife suitable for the target players.

The actual realization of the game, so as the parts strictly related to graphics, sounds and user interface, are still under development.

V. Technical Aspects

In our project we use Unity 3D. The Unity scripting system relies on the Mono implementation of the .NET framework. This allows the game developer to use in the game any software library compatible with the .NET framework.

Our library, in order to be compatible in Unity, has been implemented on the .NET framework, using the C# language. For this reason, it can be used also with other engines based on the .NET platform, like Paradox[3] or WAVE[4]. We found

[3]http://paradox3d.net/
[4]http://waveengine.net/

that Unity 3D was the optimal choice for our application, since it is the most used game engine worldwide, with more than 3.3 million registered developers and a 45% market share[5]; moreover, it supports all the major platforms (including mobile ones like Android and iOS). From the development point of view, it allows rapid prototyping of games with minimum effort and provides a fully integrated development environment, with a visual editor.

The interpreter part of the library uses the DotNetRDF[6] API to read the ontology in OWL language and initialize the data structure used by the semantic engine. DotNetRDF is however still limited in reasoning capabilities, because it offers only RDFS reasoning and a generic rule reasoner; on the contrary, many semantic reasoners are implemented in the Java language (e.g. Pellet[7] and Hermit[8]).

Since performing semantic reasoning on the knowledge base is needed, we have developed a Java tool that loads any number of ontologies, performs inference on the resulting knowledge base using the Pellet reasoner, and produces a single "result" file, serialized in RDF/XML notation. This file becomes the input of the semantic layer library (fig. 5).

Performing the reasoning as a pre-process step is, in our case, the optimal solution. As the knowledge base does not change during the game, we do not need to re-calculate the inferred axioms at run-time. Thus, the time required for the automated reasoning does not affect the loading time of the game. In fact, the access to the knowledge base is no longer required, once the semantic engine has been initialized, since its entity data structure contains "working copies" of the original instances.

A final consideration is about the overall performance of the system, since the use of a semantic engine introduces an overhead in terms of computational cost. In fact, we do not update the semantic engine at each frame update of the game engine; we do it at fixed time intervals instead.

We found that setting a 0.1s delay between each update of the semantic engine does not produce any visible difference in terms of gameplay. From our preliminary tests inside Unity, the semantic engine is able to update the state of up to 200 entities, without causing a visible drop in the framerate; however, further optimizations to the semantic engine code are possible.

VI. Conclusion

In this paper we have presented a semantic layer able to manage the semantics of the game entities separately from a generic game engine. We discussed the technical issues and evidenced the advantages of this approach, among which the new opportunities for the creation and distribution of reusable components for games.

It appears natural finding similarities between the proposed approach and the techniques adopted in the field of game artificial intelligence. However, game AI aims to provide efficient solutions to more specific problems, such as modeling agent behavior (e.g. behavior trees [21] or planning strategies [22]). These issues are also considered in our framework and consequently the various game AI techniques should be rather considered a useful complement.

The semantic layer is particularly suitable for serious game design and, more in general, for edutainment applications, where the learning content can be developed and used as an off-the-shelf component rather than being adapted to each specific application. In a broader perspective, we see also promising applications for the development of entertainment games. In fact, as the graphic quality of the games increases, it is possible to perceive a gap between the visual realism of virtual objects and the limited interactions they provide to the player. This happens because specifying all the possible interactions in a hard-coded manner is a time-consuming activity; furthermore, these interactions are also difficult to maintain coherent when the game design changes.

The introduction of the semantic layer, although not solving these problems directly, allows to represent the knowledge of the game world to a higher level of abstraction, and ultimately to enable a more natural form of interaction between game entities, even in ways that the designer had not prevented initially. Some game genres are more suitable than others for this type of dynamic interaction. We like to mention some here:

- **Adventure games with randomized elements**: a graphical adventure typically features item-based puzzles. In order to advance in the game, the player has to pick up and use/combine several items in a specific sequence in order to advance (using a knife to cut a rope, for instance). However, once the game is over, the player knows the solution to all the puzzles, so he/she has no reason to play it again. A possible solution to this problem would consist in randomizing some sections of the game. Using constraint-based techniques, the game designer would specify only the final goal (i.e. obtaining a certain item, like a key), letting the game decide all the intermediate steps. A number of objects could be selected from a general repository, and subsequently placed in the game scene, with the guarantee that there is a sequence of steps involving the use of these objects which leads to the specified goal.
- **Role-playing games**: games like *The Elder Scrolls V: Skyrim*[9] implement a system of rules, which involve a variety of inter-related aspects such as skills, equipment, levels of experience, fighting. These systems have many similarities and then can be seen as variants of a unique template, which a designer could customize for a specific project.
- **Stealth-based games**: in these games, the player is rewarded by avoiding enemies rather than fighting them

[5]http://unity3d.com/public-relations
[6]http://dotnetrdf.org/
[7]http://clarkparsia.com/pellet/
[8]http://hermit-reasoner.com/

[9]http://www.elderscrolls.com/skyrim

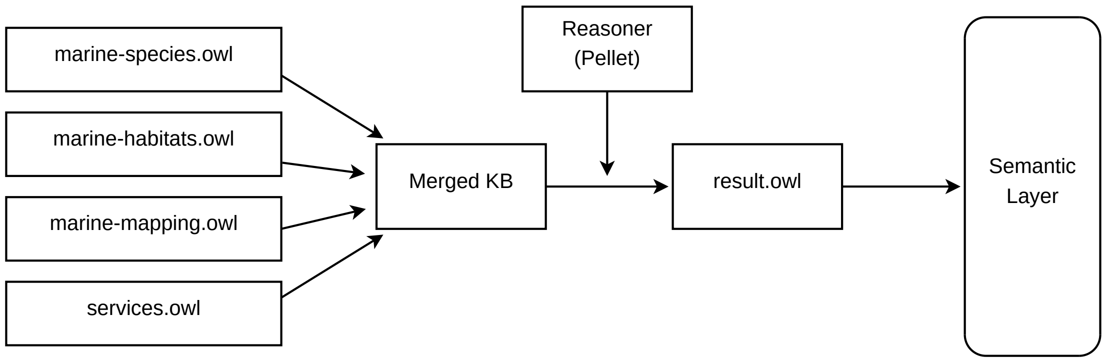


Fig. 5. The merge procedure in the case of the serious game application set in the marine environment.

directly: the *Metal Gear*[10] series represents a typical example. Enemies usually follow a patrol route that is fixed in advance by the game designer, according to the geometry of the environment. While this approach is enough for many applications, we could consider the possibility of letting the enemy figure out the best patrol route, by visiting more often the areas that are considered relevant from a security standpoint: this may depend from the objects that are placed in a certain area and consequently from the meaning of that area.

ACKNOWLEDGMENT

This work has been supported by the PO CRO Fondo Sociale Europeo Regione Liguria 2007-2013 Asse IV “Capitale Umano” ob. Specifico I/6 through the project “Serious Game per la fruizione di contenuti e l’apprendimento delle tematiche educative, culturali e ambientali in strutture museali di tipo edutainment”. The authors would like to thank Roberta Parodi (Costa Edutainment SpA) for the support in the formalization of the marine environment.

REFERENCES

[1] C. E. Catalano, M. Mortara, M. Spagnuolo, and B. Falcidieno, “Semantics and 3d media: Current issues and perspectives,” *Computers & Graphics*, vol. 35, no. 4, pp. 869–877, 2011.

[2] T. Tutenel, R. Bidarra, R. M. Smelik, and K. J. D. Kraker, “The role of semantics in games and simulations,” *Computers in Entertainment (CIE)*, vol. 6, no. 4, p. 57, 2008.

[3] T. R. Gruber, “A translation approach to portable ontology specifications,” *Knowledge acquisition*, vol. 5, no. 2, pp. 199–220, 1993, relazione marzo 2014.

[4] S. De Freitas, “Serious virtual worlds,” *A scoping guide. JISC e-Learning Programme, The Joint Information Systems Committee (JISC), UK*, 2008.

[5] T. Tutenel, R. M. Smelik, R. Bidarra, and K. J. de Kraker, “Using semantics to improve the design of game worlds.” in *AIIDE*, 2009.

[6] O. De Troyer, W. Bille, R. Romero, and P. Stuer, “On generating virtual worlds from domain ontologies,” in *In Proceedings of the 9th International Conference on Multi-Media Modeling*. Citeseer, 2003.

[7] R. M. Smelik, T. Tutenel, K. J. de Kraker, and R. Bidarra, “A declarative approach to procedural modeling of virtual worlds,” *Computers & Graphics*, vol. 35, no. 2, pp. 352–363, 2011.

[8] D. Isla and P. Gorniak, “Beyond behavior.” [Online]. Available: http://www.gdcvault.com/play/1267/(307)-Beyond-Behavior-An-Introduction

[9] M. Kallmann and D. Thalmann, “Direct 3D interaction with smart objects,” in *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, ser. VRST ’99. New York, NY, USA: ACM, 1999, pp. 124–130. [Online]. Available: http://doi.acm.org/10.1145/323663.323683

[10] C. Peters, B. McNamee, S. Dobbyn, and C. A. O’Sullivan, “Smart objects for attentive agents,” in *WCSG 2003*, 2003.

[11] T. Abaci, J. Ciger, and D. Thalmann, “Planning with smart objects,” *WSCG 2005*, p. 25, 2005.

[12] P. Sequeira, M. Vala, and A. Paiva, “What can i do with this?” in *Finding Possible Interactions between Characters and Objects. In Proc. of the 6th Int. Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 07)*, 2007, pp. 14–18.

[13] M. Badawi and S. Donikian, “Autonomous agents interacting with their virtual environment through synoptic objects,” *CASA 2004*, pp. 179–187, 2004.

[14] J. J. Gibson, *The ecological approach to visual perception*. Psychology Press, 1986.

[15] J.-L. Lugrin, M. Cavazza, S. Crooks, and M. Palmer, “Artificial intelligence-mediated interaction in virtual reality art,” *Intelligent Systems, IEEE*, vol. 21, no. 5, pp. 54–62, 2006.

[16] J. Kessing, T. Tutenel, and R. Bidarra, “Services in game worlds: A semantic approach to improve object interaction,” in *Entertainment Computing–ICEC 2009*. Springer, 2009, pp. 276–281.

[17] G. A. Miller, “Wordnet: a lexical database for english,” *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[18] J. Kessing, “Services in game worlds: A semantic approach to improve object interaction,” Master’s thesis, Delft University of Technology, Delft, the Netherlands, 2009. [Online]. Available: http://graphics.tudelft.nl/Publications-new/2009/Kes09

[19] J. Kessing, T. Tutenel, and R. Bidarra, “Designing semantic game worlds,” in *Proceedings of the The third workshop on Procedural Content Generation in Games*. ACM, 2012, p. 2.

[20] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean, “Swrl: A semantic web rule language combining owl and ruleml,” 05 2004, 21. [Online]. Available: http://www.w3.org/Submission/SWRL/

[21] D. Isla, “Handling complexity in the Halo 2 AI,” in *Game Developers Conference*, vol. 12, 2005.

[22] J. Orkin, “Three states and a plan: the AI of FEAR,” in *Game Developers Conference*, vol. 2006. Citeseer, 2006, p. 4.

[10]https://www.konami.com/mgs/