Automatic Customization of Non-Player Characters using Players Temperament *

Hector Gómez-Gauchía and Federico Peinado

Depto. Ingeniería del Software e Inteligencia Artificial
Universidad Complutense de Madrid, Spain
hector@sip.ucm.es, email@federicopeinado.com

Abstract. Believability is a basic requirement for non-player characters of videogames. Players enjoy characters with personalities that reflect human behavior, specially if those personalities combine well with players' temperaments. This paper explains a model for customizing automatically non-player characters (NPC) according to the players temperament, which is obtained before the game session. The model uses Case-Based Reasoning and Ontologies to adapt the behavior of a NPC, which is the companion of a player character in the described example.

Keywords. Player Modeling, Narrative Environments, Interactive Digital Storytelling in Entertainment, Affective Computing

1 Introduction

Videogames are pieces of procedural art. The goal of this art is evoking emotion in the player through performance of tasks in a fantasy world. There are many possible ways of evoking emotion, but it is well known that each player is emotionally affected in a particular way, depending on how the tasks of the game relate with his temperament.

In section 2 part of the state-of-art in game customization is presented. Research on game adaptability is significantly less developed than the same field of hypermedia or intelligent tutoring systems [2]. Usually current work is usually more related to the automatic adjustment of the difficulty level of action or strategy games, while this paper focus on Role-Playing Games (RPGs), characterized by strong emphasis on social interaction with Non-Player Characters (NPCs) as part of the in-game tasks instead of combat or management of resources.

NPCs are characters of the game which are controlled by the system and which are able to interact with the player's avatar interchanging items and information. The scenario of section 3 represents a short dialogue with an NPC. The model, explained in sections 4 and 5, is illustrated with this scenario because conversation is very important in the game experience of RPGs.

* Supported by the Spanish Ministry of Education and Science (TIN2005-09382-C02-01, TIN2006-15140-C03-02 and TIN2006-14433-C02-01 projects), Complutense University of Madrid and the G.D. of Universities and Research of the Community of Madrid (UCM-CAM-910494 research group grant).

Players enjoy when NPCs behave as they expect and these expectations are projections of their temperaments. We represent temperaments as a set of static values established for the current player before the game session by a questionnaire following the Keirsey's theory [6], described in section 4. The model has a knowledge base with different versions of the dialogue with an NPC related to different combinations of temperament values. Usually, each player has a new combination of those values, so a particular version of the dialogue has to be generated automatically using the knowledge base and a set of *variations* of the behavior of the NPC.

2 Current Approaches to Game Customization

Traditionally, videogames have hard-wired character behavior. Even RPGs or graphic adventures offered reasonable but simple conversation trees or state-based interaction with NPCs. New titles "sell" exciting features of their NPCs as improved intelligence, emotion or autonomy but few games are really adaptable to each particular player without some manual configuration. Blade Runner is a classical example of re-playable game whose NPCs do not act the same way each time the game is played; but the variations do not adapt the content of the game to the characteristics of the players.

A well-known example of automatically customizable game is Max Payne. Most of games have just a short number of difficulty levels that has to be manually selected by the player. In Max Pay, an auto-dynamic difficulty adjustment mechanism is implemented, allowing a more adaptable game experience to the skills of the player, but not to his temperament.

Fable uses a moral approach for the automatic customization of game content. The actions of the players can be considered good or bad by the system, and, as a consequence, their reputations change during the game, affecting the behaviour of NPCs as well. It is an interesting approach but customized only for the black-or-white fictional temperament of the player's avatar, not for the player himself.

There are many research projects trying to adapt game contents according to the feedback of the player. From the point of view of social interaction, Façade [7] is a good example. The NPCs of this interactive drama are able to speak, express emotions and even understand what the player is saying in a simplified English language. The games chooses different endings based on decisions taken by the player's avatar, but the game does not model the player's temperament.

Instead of developing an standalone system, it is becoming popular the use of middleware for the dynamic connection of external intelligent systems with commercial game engines to modify the behavior of NPCs and other game functionalities. The automatic NPC customization system could be implemented specializing some of these software platforms as Zocalo[1], based on Web Services

[1] http://zocalo.csc.ncsu.edu/

about Planning, I-Storytelling[2], based on Hierarchical Task Networks or KIIDS[3] based on Case-Based Reasoning and Ontologies.

3 The Example Scenario

The automatic NPC customization model presented here is generic, so it needs to be instantiated for a particular application involving some virtual environment, NPCs and players such as a commercial videogame.

This example scenario is an independent adventure implemented as a simple module of Neverwinter Nights[4]. The player plays the role of Drax, a knight returning home after a battle against the forces of Evil. When Drax enters his castle, one servant – a NPC – is waiting for him. The interaction with this NPC is just a short introduction to the adventure of the module that the model is trying to customize. Usually short conversations with secondary characters are not relevant for the global game experience, but this example is useful enough to show how the model works.

In Figure 1, we show an informal version of the scenario dialogue. Besides the canned text of dialogues, we focus on other features which change during the customization process. There are also different NPC animations in each dialogue mode. By default the servant falls to the floor and performs fast and repeated movements of worship to his lord. That is considered a funny animation for a fantasy game like this, but probably it is not the most appropriated animation from the point of view of a player with a serious temperament –this gesture could be consider grotesque or even offensive by some cultures or religions–.

4 Elements of the Automatic NPC Customization Model

Our line of work during the last years has been the research of different techniques and approaches to build Knowledge-Intensive Case-Based Reasoning (KI-CBR) systems, i.e.: integrated Knowledge Based Systems that combine case specific knowledge with models of general terminological domain knowledge [3, 4]. Like in previous works we use Description Logics (DLs) based languages that are commonly used to implement ontologies, and have been proven to be useful to formalize aspects of representation and reasoning in CBR systems [8, 3]. We use ontologies to represent the explicit knowledge of the model: the temperament theory, the possible variations to customize the game and the player's profile which includes his own temperament and personal data.

In Figure 2 we describe the main elements of the proposed model. There is a player playing the game, which has a standard NPCs behavior. A CBR task generates an automatic NPCs behavior customization, which consists in a set of

[2] http://www-scm.tees.ac.uk/users/f.charles/
[3] http://federicopeinado.com/projects/kiids/
[4] http://nwn.bioware.com/

Servant: Welcome home, sir. [Repeating fast worship movements]
Drax: Hum, I feel so tired... Please, servant, prepare the bath.
Servant: Immediately, sir. [Running to the bathroom]

Fig. 1. Example dialogue at the 13th level of politeness

actions performed using native commands of the game or scripts developed ad-hoc. After this customization the player finds NPCs behavior more similar to his temperament. The overall reasoning task follows this cycle: the model retrieves a case with the most similar temperament to that of the player. Inside the case there is a set of game variations to customize it to player's temperament. If the distance between the retrieved case is bigger than a specific threshold, the model adapts the variations using the distance. The player profile stores all this information to use it every time that specific player plays again. The ontologies are describen in next subsections.

From a practical point of view, Neverwinter Script implements the plain interaction – without temperament-dependent modifications – using the Aurora toolset which is included in Neverwinter Nights.

The case base has cases which contain pairs of temperament-behavior change. If the temperament of the current player is exactly the same as one in the case base, the corresponding behavior change is performed; otherwise the system search for a similar temperament in the case base and the corresponding behavior is used as a basis to create a more suitable one.

This model relies heavily on ontologies. The terms of an ontology describe the static knowledge of each aspect of the design. Ontology is a term borrowed from philosophy that refers to the science of describing the kinds of entities in the world and how they relate each other. Ontologies based on DLs paradigm include definitions of concepts –classes–, roles –properties– and individuals. Given such an ontology, the formal semantics of the language that we use specifies how to derive its logical consequences, i.e. facts not literally present in the ontology, but entailed by the semantics. To formalize our ontologies we use the DL-specific

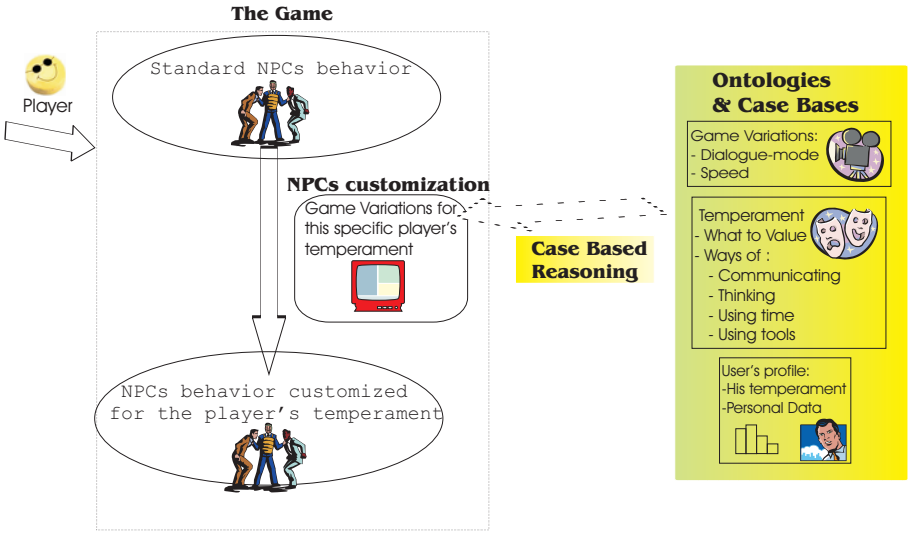

Fig. 2. Elements of the automatic NPC customization model

part of the Ontology Web Language[5] (OWL DL). These entailments may be based on a single document or multiple distributed documents that we combine using the import OWL mechanisms. The OWL DL reasoning capabilities relies on the good computational properties of DLs.

We define cases as individuals of a concept which belongs to the ontology. A case is a complex individual that has several slots and facets represented here as properties and data type properties. Each property may contain individuals of other concepts of the same ontology or other imported ontologies. The advantage of this approach is that a reasoner may check the consistency and may classify the kind of individuals automatically. We use very narrow ontologies and cases to simplify the design and updates. This model is embedded in another model [5], and both share the ontologies and case bases.

Temperaments: TEMPOnto and the Case Base We use temperament theory of David Keirsey [6], which is widely applied in psychology and in companies to interview job candidates. Keirsey's theory is centered in the long-term behavior patterns, i.e., what people do. It is an interpretation of the Myers-Briggs and Carl Jung's writings on personality types, more interested in what people think. We consider Keirsey's model more relevant for videogames because acting in them is usually more important than thinking. The theory defines four basic temperaments. Temperaments are not variable as emotions or feelings. Each person has a unique proportional combination of the four temperament types.

In this article we use an example where we consider an unique proportional combination as the description of a new case: *Artisan 10%, Guardian 10%, Idealist 30% and Rational 50%.*

[5] http://www.w3.org/TR/owl-guide/

Normally one of the temperaments is predominant, *Rational* in the example. This means that the person will behave most of the time like that temperament.

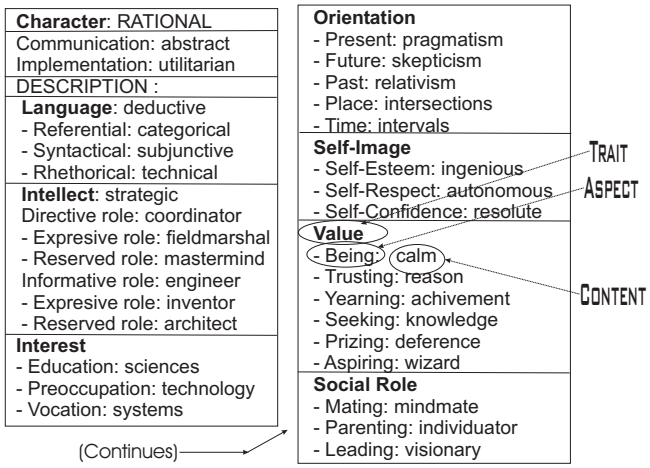

Fig. 3. The basic temperament Rational and its traits

We describe each temperament type by a set of traits composed by some aspects. In Figure 3 there is a complete set of traits for the Rational temperament. Each trait characterizes the way people behave in relation with its set of aspects.

For our example we consider the trait "value", that means what people will value most in relation with the different aspects of that trait. The aspects of the "value" trait are shown in Figure 3. One of them is *being*, that means how people appreciate to be. Rational people value being *calm*. The word "calm" is the value of the aspect "being". To represent it in the ontology, shown partially in Figure 5, we create one individual *temp:ValueBeingCalmX* as instance of *Trait*.

In contrast with the previous example, Artisan people value most being *excited*. Another example is the trait *Language* which has a *rhetorical* aspect. Rational people use a *heterodox* kind of rhetorical aspect. In contrast, Artisans prefer a *technical* kind of rhetorical aspect.

We consider that if the model may use different aspects of NPCs behavior according to the player's temperament, the player will feel more comfortable and enjoy more the videogame. We personalize the game applying variations of the game, such us the speed of movements or the language of the NPCs. Each variation may affect some traits of one specific temperament.

Variations: VARIOnto and the Case Base We call *variations* to all the possible changes that we may execute to personalize a domain. For example, in the domain of our example scenario, we decide to customize the game by the implementation of the idea represented by one assertion: "a polite NPC with slow movements makes you feel calm". From this assertion we deduce that is

important to customize two aspects, the politeness of the NPC's sentences and the speed of its movements.

To customize the NPC there are two kinds of variations depending on its relation to the virtual environment we use: *native variations*, which are direct commands to execute in the game, such as changing the speed of the animation of an NPC, or *plug-in variations*, which are embedded scripts that can be executed in the game at runtime, such as changing the dialogue mode of an NPC.

The technology we use to send commands to Neverwinter Nights is Shadow Door[6], a DLL for Neverwinter Nights Extender[7] which allows external systems to send messages by sockets to the core of the game.

These messages perform *native variations* in the game or activate *plug-in variations*. Native variations are easier to implement – just changing accordingly built-in variables of the game that controls the behaviour of NPCs –. Plug-in variations are more interesting. This is the case of the "change dialogue-mode" script in our scenario. We developed that script to maintain a variable "dialogue-mode"; its values represent possible dialogues of the NPC, i.e.: more or less polite, funny, aggressive, etc. These variations relate with the different temperaments. For each variation there are several degrees of intensity in an ordered list. For instance, the 1st level of politeness is the most polite: this variation uses a set of very polite sentences that replace the complete text of the conversation between Drax and his servant.

Figure 5 illustrates the process of implementing the assertion. In the ontology is the variation concept, which has several properties. The values of a property are individuals of other concepts. We create a variation, the NPCcalmProducer1 instance of *NPCcalmProducer*. The main properties of a variation are:

- *affectToTraits* has several traits of temperaments which are affected by the variation. Our variation example has the *temp:ValueBeingCalmX* trait.
- *hasExecutionSteps* are the necessary actions to execute the variation. In our example we have two actions, "to change the NPC dialogue mode to politeLevel-8" and "to change the NPC speed to 0.75". To implement these actions we use commands called *ExecutionSteps*.
  - `change!#dialogue-mode#politeLevel-8`, which is implemented by the individual execStepChangeDialoguePolite of the *ExecutionStep* concept.
  - `change!#speed#0.75` which is implemented by the individual execStepChange-SpeedSlow of the same concept.

Each *ExecutionStep* individual has two main properties:

- *hasActivationCommand* has the command to execute the step, e.g.: "change!" in our example. For the speed command, "change!" corresponds to a native command of Neverwinter Script which changes the speed of the NPC behavior. For the dialogue-mode command, "change!" corresponds to a script implemented specifically by the customization developers for choosing different conversations trees.

[6] http://www.cs.northwestern.edu/ rob/software/shadow%20door/
[7] http://nwnx.org/

– *hasActivationParameters* the parameters of the command. The parameters are of the ExecParam concept that is explained below.

The *ExecParam* concept is specially important for the adaptation of cases. To adapt a case we need a range of flexibility. We get it by the declaration of possible ranges. Each one refers to one of the basic temperament. We continue in our example only with the ExecParamSetDialoguePolite, because execParamSetSpeedSlow is too simple to illustrate the whole adaptation mechanism. The main properties of the *ExecParam* concept are:

– *hasExecParamName* has the literal name of the parameter to be executed by the command. In the example they are "dialogue-mode" and "speed".
– *hasExecParamValue* has the value of the parameter. In the example they are "politeLevel-8" and "0.75".
– *possibleAdaptationRange* has four subproperties: idealistRange, artisanRange, rationalRange, guardianRange. It indicates how strong is the effect for each basic temperament Each one has a list of values. They describe the distance of the value to that specific temperament. The first value in the list is the "nearest" for that temperament, i.e.: with the strongest effect, and the last value is the "farthest" for that temperament, i.e.: with the weakest effect. An example of these lists is in Figure 4.
– *hasExecParamRelevancy* indicates the general relative importance in the adaptation process. It contains the corresponding four subproperties. Each subproperty is the relative importance specifically for each temperament. For example, a parameter has a very low value in guardianRelevancy because it is very little related with that temperament, e.g.: the politeness is less relevant for Guardian people than for Rational people.

Mapping Temperaments into Variations As we mentioned before, each variation may affect some traits of a specific temperament. In Figure 5 we depict the mechanism to represent this process: each variation is related to specific traits and values of temperaments, not just with the temperament itself. This relation is performed using affectToTraits property of *Variation* concept, which contains the affected traits of *Temperament* concept. In affectToTraits property of our example we have *ValueBeingCalmX* which is of the concept of *value* trait for Rational temperament. The way to promote this calm situation is with the execution steps: execStepChangeDialoguePolite and execStepChangeSpeedSlow.

Other important aspect to represent in the model is that each variation affects in a different way to each temperament. In our example, the dialogue mode of the NPCs affects *value* trait of Rational temperament in a very different way to the same trait of Artisan temperament. This is because the first one has "calm" as value in the *value* trait and the latter has "excited". The mechanism to represent the different effects in both temperaments is through – see Figure 4– the execution parameters contents of the execution steps belonging to the variation. We describe how we adapt the contents in section 5.

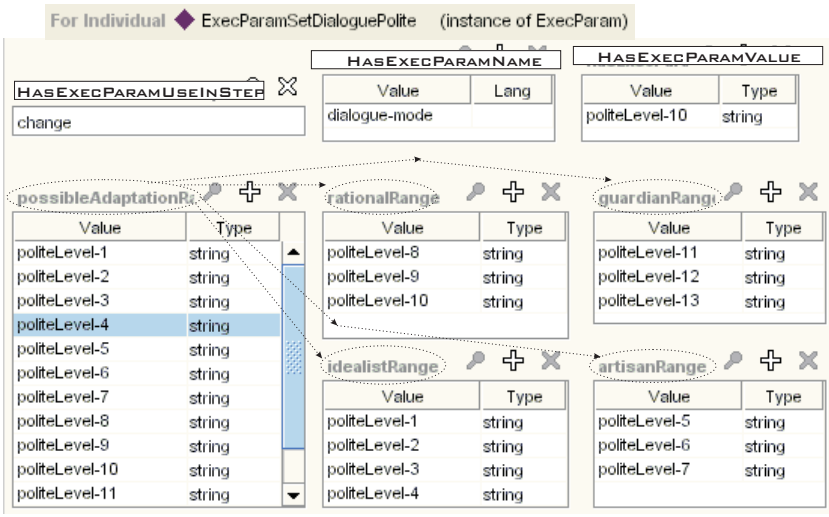

Fig. 4. ExecParam of a variation concept and ranges to be adapted

User Types and Users: USEROnto and the Case Base In the USEROnto there are two kinds of cases, the UserType and the User. The *User case* describes the player knowledge needed by the model and has these main properties:

- *hasTemperamentProportions* of the four basic temperaments in %.
- *hasUserType* that is the best match of player types.
- *hasUserAdaptations* with adaptations of that UserType to the specific player.

The property *hasUserAdaptations* is empty when the UserType matches exactly or very near with the player's temperament proportions. The adaptations are obtained from the ranges of each variation explained in VARIOnto.

The *UserType case* represents the unique proportion of the four basic temperaments and the variations generated and stored for it. Each case in the case base is one of these combinations. To avoid a huge number of cases, a minimum gap between cases is defined a priori. The name of the example UserType case is *UserType-A10-G10-I30-R50*. The main properties are:

- *hasTemperamentProportions* is the same property as in the player case.
- *hasUserType* with the variations for this player type.

5 Process of the Automatic NPC Customization Model

We use the previous example to illustrate the general reasoning cycle in our model that follows the classic CBR cycle [1]. We query the CBR system with a description based on the result of the Keirsey's questionnaire. Suppose the form gives us the following player temperament proportions: Artisan 10%, Guardian 10%, Idealist 30% and Rational 50%. The reasoning cycle *retrieves* the most similar case (e.g. Artisan 10%, Guardian 30%, Idealist 30% and Rational 30%).

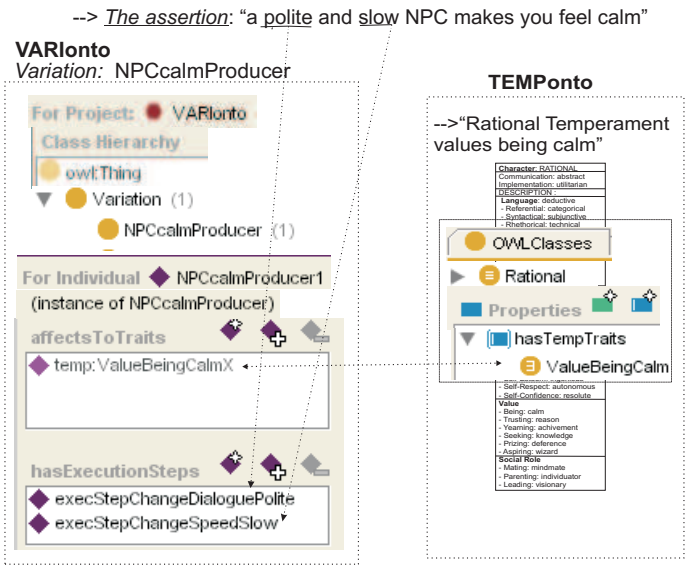

Fig. 5. An assertion as a *variation* related to a specific *trait* of a *temperament*

To measure the similarity between cases we use the following definition of distance which is a sum of the differences for each of the four temperaments:

$$\sum_{T\text{=}Artisan}^{T=Rational} (\%NewCase_T - \%retrievedCase_T) * CorrectionFactor_T$$

The *%NewCase* is the proportions of the query. There is a correction factor that is proportional to the previous difference. The correction factor is very high when the retrieved case is very far for one of the temperaments with a high percentage, because this retrieved case is not a good case even if the other percentage temperaments are similar.

The system *reuses* the retrieved case by adapting the retrieved case output slots, i.e.: a set of variations for each of the four basic temperaments. Variations are in its own case base. The adaptation actions are calculated by the difference between the proportions in the query and in the retrieved case; in our example the adaptation actions are: Artisan stays unmodified, make Guardian decreasing 20%, make Rational increasing 20% and let Idealist without modifications. After the adaptations, the system executes the modified variations. These variations represent the solved case.

The system performs the adaptation of each variation using the *possibleAdaptationRange* property that has the possible ranges for each temperament as described before. Figure 4 is our variation example; we see that the range for each temperament is a list of values, e.g.: *rationalRange*. The total number of values (e.g.: 3) is equivalent to 100%. To make the retrieved case 20% more Rational we calculate that proportion in the list and we choose the element which occupies

the position corresponding to the obtained proportion. The current mood filters the adaptation, increasing or decreasing some of these adaptations.

Next tasks that are under development are *revision* of the solved case using the feedback of the player after the game session, and *remembering* of the useful cases to be reused in further iterations.

Servant: Welcome to the castle, sir! Everybody has heard about your courage in the battlefield.
Drax: I am tired because of the battle, servant. Prepare my bath.

Servant: At your command, my lord. [Saluting him] A perfumed bath is ready for you.

Fig. 6. Example dialogue at the 8th level of politeness

In Figure 6 the customized dialogue is shown. For the same scenario of Figure 1 the player obtains a different conversation because the system is aware of her temperament and therefore of the preferences on NPCs. That is the reason why the politeness of the conversation has increased from the 13th to the 8th position. The speed of the movement decreased proportionally, but it cannot be appreciated in a screenshot.

6 Conclusions

We present a model for customizing automatically NPCs according to the player's temperament. Using Ontologies and CBR the system modifies the animations of two characters and their dialogue in the context of a fantasy RPG.

The development is not mature enough for being formally evaluated, but some limitations arise from a practical point of view: e.g. an out-of-game questionnaire could not be the most comfortable way to identify the real temperament of the player. The case base of mappings between temperaments and NPC behaviors is time consuming because it has to be refined based on the experience of players; this refinement applies too to the implementation of each NPC behaviour, defined in abstract terms, for a specific platform as Neverwinter Nights

or other videogame engine. Another limitation is the difficult integration with other architectures of autonomous characters whose personality may eventually contradict the behavior recommended by the system.

Besides the areas mentioned in the example, there are other areas of customization for NPC behavior, such as abilities, appearance (race, clothes, etc.), social role, etc. The same approach can be used for customizing those features. Also it can be used with any number of secondary characters as villains, partners, lovers, etc. Every element in a game is susceptible of being customized according to temperaments of players, as the weather, ambient light, camera settings, etc.

Modelling artificial temperaments of NPCs is also possible using the same theory presented here or other temperament theories. Such application could let the system customize whole groups of characters as families, troops, or even the whole cast of the game using a compatibility table of temperaments.

The process of customization according to different player temperaments is independent of the game domain and genre, except in the set of variations that have to be specifically implemented for the game (VARIOnto).

Currently the temperament is a static value in the model taken from the results of a questionnaire, but a future line of research will be to identify dynamically other values (e.g. player emotions) according to the actions of the avatar using specific heuristics.

References

1. E. Armengol and E. Plaza. A knowledge level model of knowledge based reasoning. In S. Wess, K. D. Althoff, and M. M. Richter, editors, *Proceedings of the 1st European Workshop on Topics in Case-Based Reasoning, Kaiserslautern, Germany - EWCBR'94*, pages 53–64. Springer–Verlag, Berlin, 1994.
2. P. Brusilovsky. Methods and techniques of adaptive hypermedia. *User Modeling and User Adapted Interaction: Special issue on adaptive hypertext and hypermedia*, 6(2-3):87–129, 1996.
3. B. Díaz-Agudo and P. A. González-Calero. An architecture for knowledge intensive CBR systems. In E. Blanzieri and L. Portinale, editors, *Advances in Case-Based Reasoning – (EWCBR'00)*. Springer-Verlag, Berlin Heidelberg New York, 2000.
4. B. Díaz-Agudo and P. A. González-Calero. Knowledge intensive cbr through ontologies. *Expert Update*, 2003.
5. H. Gómez-Gauchía, B. Díaz-Agudo, and P. A. González-Calero. Cobber, toward an affective conversational ki-cbr framework. In B. Prasad, editor, *B. Prasad(Ed.) Procs of the 2nd Indian International Conference on Artificial Intelligence IICAI-05*, pages 1804–1820, Pune, India, December, 20-22 2005. IICAI.
6. D. Keirsey. *Please Understand Me II.* Prometheus Nemesis Book Co Inc., 1998.
7. M. Mateas and A. Stern. Façade: An experiment in building a fully-realized interactive drama. In *Game Developer's Conference, Game Design track*, San Jose, California, USA, 2003.
8. S. Salotti and V. Ventos. Study and formalization of a case-based reasoning system using a description logic. In B. Smyth and P. Cunningham, editors, *Advances in Case-Based Reasoning – (EWCBR'98)*. Springer-Verlag, 1998.