Automatic Personalization of the Human Computer Interaction Using Temperaments [*]

Hector Gómez-Gauchía, Belén Díaz-Agudo and **Pedro A. González-Calero**
Dep. Sistemas Informáticos y Programación
Universidad Complutense de Madrid
{hector,belend,pedro}@sip.ucm.es

Abstract

In this paper we model a personalization system which takes into account the user interaction styles regarding a software artifact and the user temperament. We propose the use of Knowledge Intensive CBR where there are cases that represent specific variations of a given software artifact, and there are ontologies for the static knowledge. Our model is generic and reusable. In this paper we exemplify it with a system to personalize the Linux operating system environment.

Introduction

There are many efforts in research and industry to design friendly GUI's. But what is the meaning of friendly? It is the same meaning for all kind of users? Obviously not. But how can the different perceptions of the users be determined? A solution may be the set of characteristics which users show when they interact with the computers. These characteristics are the result of the user's temperament.

The concept of Interaction Styles refers to all the ways the user can communicate or otherwise interact with the computer system. The concept belongs to the realm of Human Computer Interaction (HCI). In HCI textbooks, such as Shneiderman (Shneiderman 1997) and Preece et al. (Preece 1994), the types of interaction styles mentioned are usually in relation to the different computer interfaces: command language, form filling, menu selection, and direct manipulation. The Interaction Styles are influenced directly by the user's temperament, there are studies that define these relations (Berens 2001).

From these studies we can extract what may appeal as friendly and comfortable for users with a specific temperament. To be able to do this we needed to build a very flexible model where we can test many refinements in order to find out the adequate conditions for each temperament. We found the flexibility in the knowledge based systems, where the conditions were independent to the software. The model we are thinking about requires a lot of general knowledge to classify each feature and reason with it for each temperament. The knowledge intensive Case-Based Reasoning (KI-CBR) has these characteristics (Aamodt 1990; Díaz-Agudo & González-Calero 2000). This approach uses ontologies, which allows us to perform the mentioned tests with flexibility and reasoning about the topics that participate in the temperaments and in the possible variations of the system. These variations represent the different aspects of a certain system that are adequate to be personalized.

The model we present is a personalization system independent of the specific software artifact. To apply it to a certain software, we only have to create the specific variations to populate the case base. In this article the domain of our example is to personalize the Linux operating system in general. To do so there is an ontology and a case base that describe the specific variations of Linux.

In the next section we describe the three main components of our model: temperaments, variations and users. For each one of them we have formalized one ontology that is specialized in a case base. We then define how to use the mapping to define how a variation affects to a temperament. Then we describe the KI-CBR reasoning cycle and the architecture where we divide the main tasks in two sets: the Knowledge Engineer tasks and the user tasks. Before conclusions, there is a brief description of a domain case study where we are implementing a prototype of the model.

The main ontologies and case bases

This model relies heavily on ontologies. The static knowledge of each aspect of the design is described by terms in an ontology. Ontology is a term borrowed from philosophy that refers to the science of describing the kinds of entities in the world and how they are related.

To formalize our ontologies we use the Ontology Web Language (OWL). An OWL ontology may include descriptions of classes, properties and their instances. Given such an ontology, the OWL formal semantics specifies how to derive its logical consequences, i.e. facts not literally present in the ontology, but entailed by the semantics. These entailments may be based on a single document or multiple distributed documents that have been combined using defined OWL mechanisms [1]. The OWL reasoning capabilities relies on the Description Logics paradigm.

We define cases as individuals of a concept which belongs to the ontology. A case has several slots and facets, repre-

[*]Supported by the Spanish Committee of Science & Technology (TIN2005-09382-C02-01)

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

sented as properties and sub-properties. Each of them may contain individuals of other objects 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. We use very narrow ontologies and cases to simplify the design and updates. This model is embedded in another model (Gómez-Gauchía, Díaz-Agudo, & González-Calero 2005), and both share the ontologies and case bases. This is the reason because the model do not use some of the classes mentioned in the ontologies.

Temperaments: TEMPonto and the case base

We use the temperament theory (Keirsey 1998), 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. These authors were more interested in what people think. The theory defines four basic temperaments. 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 %*.

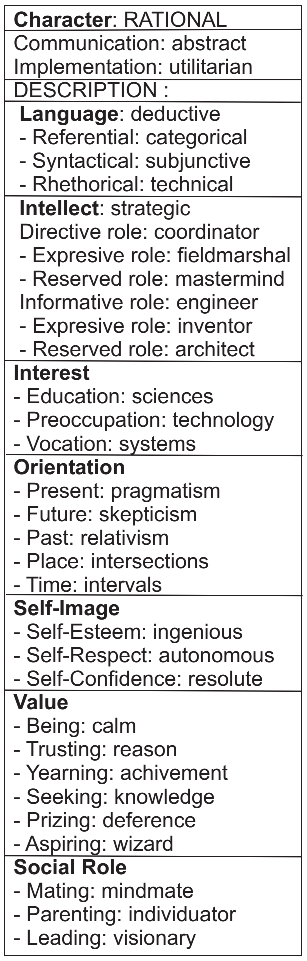
Normally one of the temperaments is predominant, *rational* in the example. This means that the person will behave most of the time like that temperament.

We describe each temperament type by a set of traits. In Figure 1 there is a complete set of traits for the rational temperament. Each trait characterizes the way people behave in relation with a particular aspect. 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 1. 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 2, we create one individual *temp:ValueBeingCalmX* of *Trait* class.

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 believe that if the model may use different aspects of language adapted to the language of the user's temperament, the user will understand better the messages and he will feel more comfortable working with the system. We personalize the system building variations which affect some specific aspects of a trait.

Variations: VARIonto and the case base

We call *variations* to all the possible actions that are executed to personalize a domain. For example, in our sample domain, the Linux system, we have the assertion "the green color makes you feel calm" which is illustrated in Figure 2. We create a variation to include this assertion as a possible personalization action. The variation is an individual GuiLookBackgroundColorTest1 of the variation subclass *GUILookBackgroundColor*. The ontology has several


Character: RATIONAL
Communication: abstract
Implementation: utilitarian
DESCRIPTION :
Language: deductive
- Referential: categorical
- Syntactical: subjunctive
- Rhethorical: technical

Intellect: strategic
Directive role: coordinator
- Expresive role: fieldmarshal
- Reserved role: mastermind

Informative role: engineer
- Expresive role: inventor
- Reserved role: architect

Interest
- Education: sciences
- Preoccupation: technology
- Vocation: systems

Orientation
- Present: pragmatism
- Future: skepticism
- Past: relativism
- Place: intersections
- Time: intervals

Self-Image
- Self-Esteem: ingenious
- Self-Respect: autonomous
- Self-Confidence: resolute

Value
- Being: calm
- Trusting: reason
- Yearning: achivement
- Seeking: knowledge
- Prizing: deference
- Aspiring: wizard

Social Role
- Mating: mindmate
- Parenting: individuator
- Leading: visionary

Figure 1: A basic temperament and its traits

classes that are necessary to define the variation class. Properties names are on top of the text boxes in Figure 2, their are in thick font in grey color. 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 only one action, "to change the background color to green". The command to execute it is:

  ```
  Gconf --set \# 008000
  ```

  The command is represented by an individual, execStepChangeBackgroundGreen, which belongs to *ExecutionStep* class (not shown in the figure.) This class has two main properties:

  – *hasActivationCommand* has the command to execute the step, e.g.: "Gconf" in our example. This is an Linux utility that changes several aspects of the GUI, depending on the parameters, that is the next property.

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

The *ExecParam* class 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. In Figure 3 it is the execParamBackgroundGreenSet individual, which is the parameter of the ExecutionStep described before. The main properties of the *ExecParam* class are (see in Figure 3, thick font and gray color):

- *hasExecParamName* has the literal name of the parameter to be executed by the command. In our example ”–set”.
- *hasExecParamValue* has the value of the param. In our example is ”#008000”.
- *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 with the strongest effect and the last value is the farthest with the weakest effect. An example of the lists is in Figure 3.
- *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 when it is very little related with that temperament, e.g.: the background color is not relevant for guardian people because they do not care much about colors. This property does not appear in the figure.

Mapping variations into temperaments

Each parameter is related with specific traits of the basic temperaments, not just with the temperament itself. This relation is by the property affectToTraits of a *variation class* that contains traits of the *temperament* class. This is depicted in Figure 2. For example, the previous example of background color affects in the *value* trait of the rational temperament in a very different way to the artisan temperament. This is because the latter has “artistic” as value in the selfEsteem trait. This indicates that the artisan people appreciate the colors from the artistic viewpoint which is very different to the guardian people that appreciate colors that produce calm. *ValueBeingCalm* is an aspect of the trait ”value” represented as a subclass of the ”value”. This is possible because the GuiLookBackgroundColorTest1 variation of our example has in affectToTraits both traits, *ValueBeingCalm* and *selfEsteemArtistic*. And the execution parameters have different ranges for each of both temperaments.

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 user knowledge needed by the model and has these main properties:

- *hasTemperamentProportions* with the percentage of each of the four basic temperaments.
- *hasUserType* that is the best match of user types.
- *hasUserAdaptations* with adaptations of that UserType to the specific user. This property is empty when the UserType Matches exactly or very near with the user’s Temperament proportions. The adaptations are obtained from the ranges of each variation that were explained in the 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 an infinite 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 user case.
- *hasUserType* with the variations for this user type.

Reasoning Cycle

We are using the previous example to illustrate the general reasoning cycle in our model (Figure 4). We follow the classic CBR cycle (Armengol & Plaza 1994). We query the CBR system with a description based on a form filled by the user. Suppose the form gives us the following user temperament proportions: Artisan 10 %, Guardian 10 %, Idealist 30 % and Rational 50 %. The reasoning cycle *retrieves* the most similar case. Let’s suppose it is Artisan 30 %, Guardian 10 %, Idealist 30 % and Rational 30 %.

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:

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

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: make Artisan decreasing 20 %, Guardian stays unmodified, 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. In Figure 3 is our variation example; we see in it that the range for each temperament is a list of values,e.g.: *rationalRange*. The total number of values is one hundred percent. To make

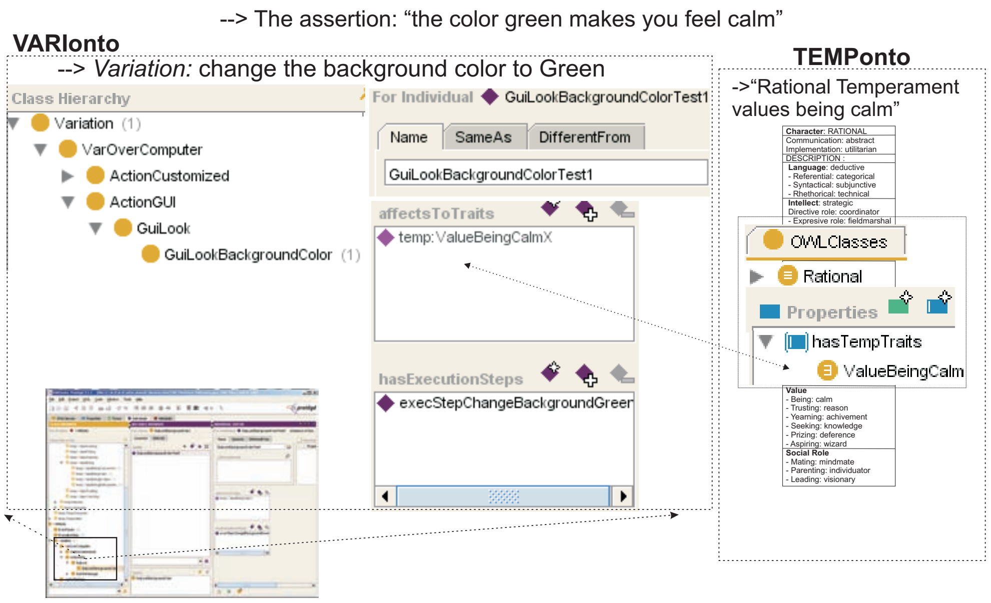


Figure 2: An assertion as a *variation* related to a specific *trait* of a *temperament*

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, i.e.: 50%. This is based in the fact described previously that the first elements in the list are nearer to the temperament, i.e.: in our example those first elements are more "rational". The current mood filters the adaptation, increasing or decreasing some of these adaptations.

The next task is to *revise* the solved case using the user's feedback. The user may agree or complain about the performed variations. If there are complains, the system repairs the case to create the repaired case.

Once the user agrees with the variations, the result is a new unique case that is *remembered*, i.e.: stored in the case base. To avoid having too many cases that are very similar each other, the store task is done only if there is at least the minimum allowed distance between the new case and the old cases. This threshold keeps a balance between having to compute a solution from scratch for each new query or to have thousands of very similar cases stored that would prevent a good case retrieval performance.

The Architecture: Main Tasks

In this section we briefly explain how we organize our model in terms of tasks, i.e., pieces of work required to be done and that are treated as basic units. As it is shown in Figure 5 our model organizes these tasks into two types: knowledge engineer and user tasks. To solve these tasks our model relies in the described ontologies, that includes the common vocabulary and general descriptions, and the DLs reasoner.

Solving the knowledge engineering tasks means defining the ontologies, making the mappings with the specific domain variations and verify them. Once these tasks are solved the reasoning cycle described previously is in charge of solving the user tasks, composing their results, and performing the corresponding variations.

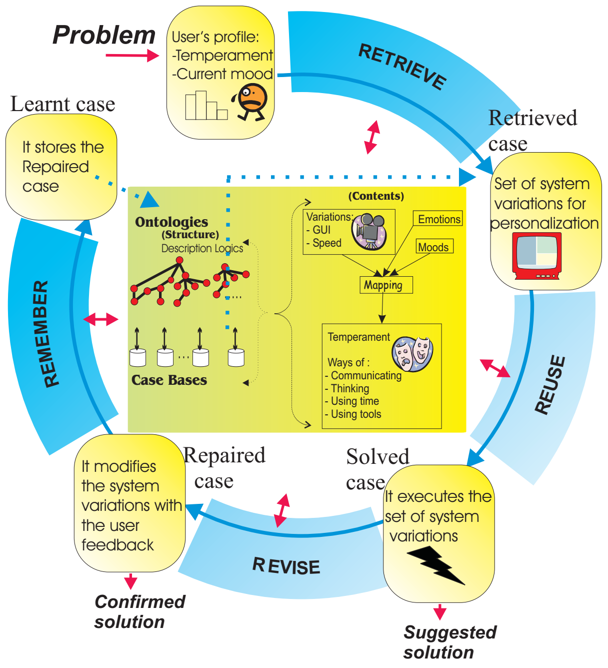
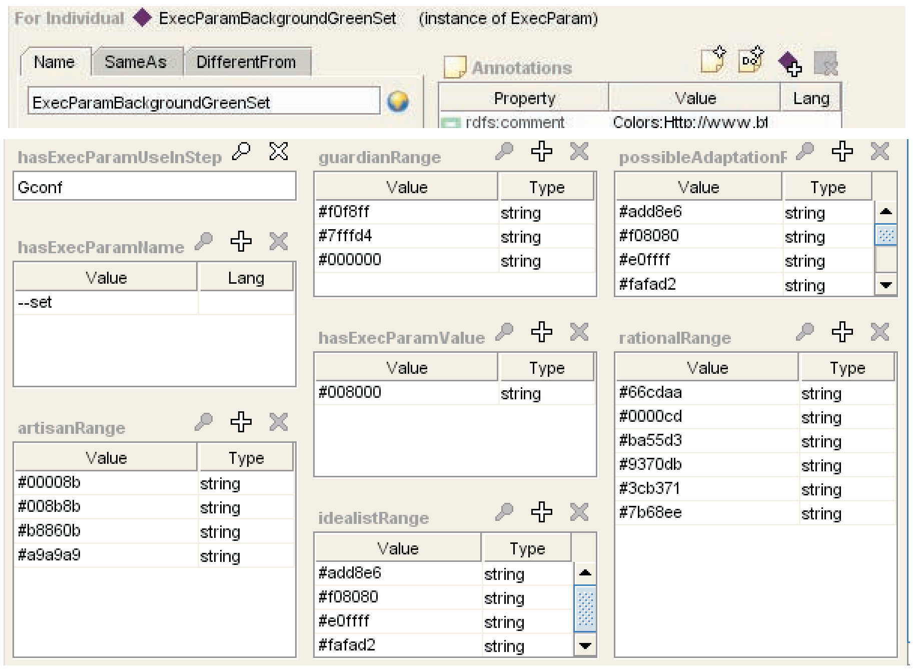


Figure 4: Overview of the reasoning cycle with ontologies

User Tasks

- **Interviewer** The first time user logs in(or when ever he wants to revise his extended profile), he fills three types of information:
  - a questionnaire. He has two options, a short questionnaire of sixteen questions or a long one of seventy questions. The first one, the so called *four type sorter*, classifies the user temperament giving a unique proportion for each of the four basic temperaments. The long one, the so called *temperament sorter II*, assigns one of the


For Individual ExecParamBackgroundGreenSet (instance of ExecParam)

Name | SameAs | DifferentFrom

ExecParamBackgroundGreenSet

Annotations

Property	Value	Lang
rdfs:comment	Colors:Http://www.bl	

hasExecParamUseInStep

Gconf

hasExecParamName

Value	Lang
--set	

artisanRange

Value	Type
#00008b	string
#008b8b	string
#b8860b	string
#a9a9a9	string

guardianRange

Value	Type
#f0f8ff	string
#7fffd4	string
#000000	string

hasExecParamValue

Value	Type
#008000	string

idealistRange

Value	Type
#add8e6	string
#f08080	string
#e0ffff	string
#fafad2	string

possibleAdaptationF

Value	Type
#add8e6	string
#f08080	string
#e0ffff	string
#fafad2	string

rationalRange

Value	Type
#66cdaa	string
#0000cd	string
#ba55d3	string
#9370db	string
#3cb371	string
#7b68ee	string

Figure 3: ExecParam of a variation class and ranges to be adapted

sixteen temperaments based on the four pairs of features represented with eight letters. The user may fill in one of the questionnaires or both. In this last situation the system has more elements to reason more accurately. In this article we focus on the model that uses the four basic temperament types.

– Background knowledge topics he fills in the extended profile are: Computer skills, Domain expertise, ...

– The last topic the interviewer asks is the current mood or emotional state. There is a set of icons reflecting several general emotional states such as happy, sad, apathetic, ... or angry. The user may choose the descriptors that better define his current mood.

– After the previous step, or when the user logs into the system for second time, it asks him to choose a mood that better fit in his current state.

The result is a query with the description of the user temperament. The personalizer uses it in the CBR cycle.

- **Personalizer** There are two types of personalization: Manual, where the user chooses the state of each of the possible variations of the system; Automatic , where the system reasons to decide the appropriate variations according to the user's temperament and his current mood. This is performed by the Case-based reasoning cycle described in the previous section. These are the functional steps:

1. The personalizer *retrieves* the most similar case to the query. The similarity is an heuristic distance between the proportions for each temperament of the query and the description of cases in the case base of users.
2. The personalizer *applies* the heuristic distance to adapt the variations of the retrieved case to the description of the query. The result is the solved case. After the adaptation, the personalizer executes the variations.
3. The personalizer *calls* the feedbacker task to revise the solved case. The user may complain about the variations supplying a set of fixes. The personalizer will modify the solved case variations using the fixes. The result is the repaired case.
4. The personalizer *stores* the repaired case, called learnt case, only if it has the minimum threshold of heuristic distance to other stored cases. If it does not reach the threshold, the personalizer stores the fixes as a particular personalization of that user.

- **Feedbacker** This task is optional. The user activates it when he disagrees with the variations. The user creates fixes, which indicate which aspects, the variation and its parameter values, he dislikes and he proposes new ones. These fixes are passed back to the personalizer.

Knowledge Engineering Tasks

- **Editor** The first task is to build the ontologies and the case bases. The knowledge engineer describes the model in terms of classes, properties and its instances. To create the case bases he uses the ontology terms. The editor uses a reasoner (Pellet[2] or Racer[3]) to verify the consistency of cases and ontologies. We use Protg as the ontology editor [4]. The other tasks use the reasoner to classify new individuals of cases and to update the ontologies and case bases.

- **Mapper** The model is independent to the personalized domain because the variations are in a separate ontology,

[2]http://www.mindswap.org/2003/pellet/index.shtml
[3]http://www.sts.tu-harburg.de/ r.f.moeller/racer/
[4]http://protege.stanford.edu/

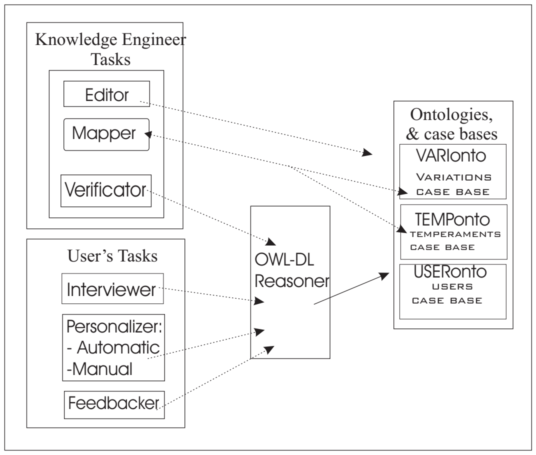


Figure 5: Tasks in the architecture of the model

VARonto. We can put variations of a new domain application in this ontology and exchange the mapping of VARonto to the temperament ontology. With only these changes the model will personalize the new domain application. This task is an editor that helps to create the relations between the variations and the traits affected by them:

– It helps to create a variation.
– It helps to create the temperaments.
– It helps to create the mapping between variation and the temp.

The most complex is to decide which traits of which temperament are affected by the variation. We found useful to ask for help to a psychologist. The positive side is that once this mapping is performed, it is not necessary to change unless a complain from the feedback is filed.

A case study: personalization of Linux

The main ontologies are created. To applied the model to the Linux system there are necessary three steps:

- To define and implement a set of variations for Linux.
- A mapping between the set of variations and the temperament ontology to indicate how each variation affects each temperament trait.
- Define a period of refinement using the feedback of the users when they do not like the personalization. Our users are a set of students of the last year of their Degree on Computer Science.

We are developing a prototype for the Debian distribution using Gnome as window manager. It allows variations at several levels of abstraction, from a very detailed level, such as the ”Gconf” utility of our example, to very generic level, such as ”themes”. Themes offer a complete unique look for the Gnome. We create a case base of themes, classify them according to the traits that affect to the temperaments and feed them as variations of the system.

Conclusions and Further Work

Since the meaning of friendly GUI is not the same for all kind of users, we propose to personalize systems based on characteristics which users show when they interact with computers. These characteristics are the result of the user's temperament.

We propose a model to personalizing software systems taking into account the user's temperament. It is independent of the domain and flexible because we use knowledge based systems to keep separated the knowledge of the domain from the reasoning process. We use the KI-CBR approach to model the static knowledge, which we represent with ontologies. The core ontologies are: variations of the system, which are the possible elements to be personalize; temperaments with traits, which are the aspects of a temperament affected by the variations; and users, which has a temperament represented by a unique proportion of the four basic temperaments.

To apply the model to a domain, i.e.: a program or operating system, is only necessary to create the specific variations to populate the case base. In this article we describe briefly a case study in the domain of Linux personalization. Currently we are developing a prototype for the Debian distribution and Gnome windows manager.

References

Aamodt, A. 1990. Knowledge intensive case-based reasoning and sustained learning. In *Proceedings of the ninth European Conference on Artificial Intelligence – (ECAI-90)*, 1–6.

Armengol, E., and Plaza, E. 1994. A knowledge level model of knowledge based reasoning. In Wess, S.; Althoff, K. D.; and Richter, M. M., eds., *Proceedings of the 1st European Workshop on Topics in Case-Based Reasoning, Kaiserslautern, Germany - EWCBR'94*. Berlin: Springer–Verlag. 53–64.

Berens, L. V. 2001. *Understanding Yourself and Others: An Introduction to Interaction Styles*. Telos Publications.

Díaz-Agudo, B., and González-Calero, P. A. 2000. An architecture for knowledge intensive CBR systems. In Blanzieri, E., and Portinale, L., eds., *Advances in Case-Based Reasoning – (EWCBR'00)*. Berlin Heidelberg New York: Springer-Verlag.

Gómez-Gauchía, H.; Díaz-Agudo, B.; and González-Calero, P. 2005. COBBER, Towards an Affective Conversational KI-CBR Framework. In *In Proceedings of 2nd Indian International Conference on Artificial Intelligence*.

Keirsey, D. 1998. *Please Understand Me II: Temperament Character Intelligence*. Prometheus Nemesis Book Company.

Preece, J. 1994. *Human-Computer Interaction*. Addison Wesley.

Shneiderman, B. 1997. *Designing the User Interface: 3 edition*. Addison Wesley.