Department of
Information Engineering
and Computer Science
DISI

UNIVERSITY
OF TRENTO - Italy

DISI - Via Sommarive 14 - 38123 Povo - Trento (Italy)
http://www.disi.unitn.it

IMPLICIT CULTURE FRAMEWORK FOR BEHAVIOR TRANSFER

Aliaksandr Birukou, Enrico Blanzieri, and Paolo Giorgini

October 2009

Technical Report # DISI-09-059

Implicit Culture Framework for behavior transfer

Aliaksandr Birukou, Enrico Blanzieri, Paolo Giorgini
DISI, University of Trento, Italy

ABSTRACT

People belong to different communities: business communities, Web 2.0 communities, just to name a few. In this chapter we show that experience acquired by people in communities constitute community culture. We introduce the problem of culture transfer between or within communities and propose a domain-independent approach for transferring community culture.

First, we formalize the notion of culture, which includes behavior, knowledge, artifacts, best practices, etc. Second, using this formalism, we propose the Implicit Culture Framework, which is an agent-based framework for transferring behavior between community members or between communities. Finally, we present and evaluate a system for web service discovery developed using the Implicit Culture Framework.

INTRODUCTION

In different areas of their lives, people form and become part of different communities. Examples include, but are not limited to business communities, hobby communities, Web 2.0 communities (e.g., in Flickr, Delicious, CiteULike, Bibsonomy), and communities of software users (e.g., BitTorrent, Firefox, OpenOffice). Such communities are often called communities of practice and are defined as ``... groups of people who share a concern or a passion for something they do and learn how to do it better as they interact regularly" (Wenger, 2009). People in a community of practice interact and develop shared competence and experience related to their activity (Wenger, 1999). The accumulated experience is probably the most important result of the community interactions and it comes in the form of behavior, best practices (Shaw & Gaines, 1999), ways of using community artifacts (Lave & Wenger, 1991) and addressing recurring problems (Wenger, 2009), and implicit or explicit knowledge (Baumard, 1999; Nonaka & Takeuchi, 1995). In other words, we can speak about the culture developed by a community.

Information about the culture can be used for improving the state of affairs of the community, e.g. by providing economic and social benefits to community members. For example, we can use the culture to facilitate the integration of newcomers into the community; to transfer and share knowledge, behavior and experience within or between communities; to discover and characterize communities.

However, a substantial part of the community culture is implicit, i.e. not readily available to all community members, even though sometimes accessible by single individuals. Still, in many cases the culture should be preserved even if the community changes. Thus, the problem of dealing with culture of communities can be formulated in terms of discovering, representing, transferring, and preserving culture. Instances of this problem are described in the literature as transfer of knowledge and retention of experience in organizations (Bender & Fish, 2001), leveraging company's knowledge (Nonaka & Takeuchi, 1995), sharing implicit knowledge in communities of practice (Gongla & Rizzuto, 2001; Mimnagh & Murphy, 2004).

Different approaches address some aspects of the above-mentioned problem. Nonaka and Takeuchi (1995) highlight the importance of knowledge for the organizations and propose a theoretical framework for knowledge creation. The framework implements the resource-based approach and

describes elements of knowledge creation and their interactions that lead to creating new knowledge. Another approach is legitimate peripheral participation, i.e. actively involving newcomers into social practices of communities. It is proposed by Lave and Wenger (1991) as an approach that facilitates acquiring of existing socio-cultural practices by new community members. In computer science, examples include recommending friends and communities in Facebook and LinkedIn, using forums, blogs, FAQ lists. There are also social navigation systems that help communities to share their experience in web search (Smyth et al., 2005), in using educational resources (Brusilovsky et al., 2005, Farzan & Brusilovsky, 2007), etc.

We argue that a more systematic computer science approach that includes engineering aspects is required to capture, represent, make explicit, and transfer elements of culture. As a result, communities will get more economic and social benefits from the use of their culture.

In order to propose such an approach, we first formalize the notion of a community culture and define it as a set of traits that are shared by the community and are transmitted. The transmission dimension points to a way of spreading culture. The sharing dimension is required for two reasons: (1) to go from the set of personal traits of an individual to the culture of a community, and (2) to filter out traits which only pertain to the community as a whole, but not to individuals. Examples of latter traits include marriage habits and birth rate.

Second, we focus on behavior as an important aspect of culture and propose the Implicit Culture Framework, an agent-based framework for transferring behavior between community members or between communities.

Finally, we show how to apply the proposed solution in the domain of recommendation systems using a system for web service discovery as a case study.

The objective of this chapter is to consolidate the effort and present a complete overview of the Implicit Culture Framework. More specific objectives are: to propose a definition of a community culture, to propose an engineering approach, the Implicit Culture Framework, for discovering, representing, transferring, and preserving community culture, and to show how to apply this approach in practice. With respect to the book's focus on culture-aware information technology, this chapter describes a framework for modeling and transferring culture in social software.

IMPLICIT CULTURE. FORMAL DEFINITION

Consistently with AI literature, we define an *agent* as a "[…] physical or virtual entity that can act, perceive its environment (in a partial way) and communicate with others, is autonomous and has skills to achieve its goals and tendencies […]" (Ferber, 1999). An agent can represent an individual or a collective entity such as an organization, and can have different *cultural traits*, which are characteristics of human societies that are potentially transmitted by non-genetic means and can be owned by an agent. The requirement ``can be owned by", which we add to the definition by Mulder (2006), means that it is possible for an agent to have a cultural trait. As we mentioned previously, different kinds of behavior, beliefs, knowledge are particular kinds of cultural traits.

To model changes in the set of traits of an agent and consequently, changes in culture, we use the notion of state. We assume that the world can be in different states and the set of traits of the same agent can be different in different states.

Let us consider the set of agents Ag, the set of traits T and the set of states S. Given an agent $a \in Ag$ and a state $s \in S$, we denote the set of traits of the agent a in the state s with $T_a(s) = \{\tau_i\} \subseteq T$ and we use the predicate $has(a, \tau, s)$ to represent the fact that the agent a has a trait $\tau \in T_a(s)$ in the state s. In the following, we call the set of traits of an individual *the culture of an individual*.

Example. Let us consider a set of people and model them as agents with a set of traits and some behavior related to transmission, in particular, *telling* and *memorizing*. Let Ag in our example is a set of people: Charlie, Pedro, Maria, and Andrea are European citizens, and Toru is from Japan. Let T be

a set of traits of different types, as shown in Table 1. For each trait, we also put its abbreviation (used in the figures later) in parentheses.

trait type	**Traits**
knowledge	Dante_Alighieri_wrote_Divine_Comedy(DA), cappuccino_is_coffee(CI), latte_macchiato_is_coffee(LM), Meiji_era_was_in_1868_1912(ME)
behavior	eating_with_sticks(ES), telling, memorizing, eating_with_fork(EF)
norms	never_put_mayonnaise_on_pizza(NP), never_open_umbrella_inside_building(NO)
beliefs	Christianity(Chr), Buddhism(Bud)

Table 1. The set of traits T in the running example.

Table 2 lists the sets of traits of the specific agents of Ag={Charlie, Pedro, Toru, Maria, Andrea}. *We can write* has(Maria, Dante_Alighieri_wrote_Divine_Comedy, s_1), *or* has(Charlie, cappuccino_is_coffee, s_1), *but not* has(Andrea, eating_with_sticks, s_1). *We will use this example as a running example.*

Set	**Traits**
$T_{Charlie}(s_1)$	Dante_Alighieri_wrote_Divine_Comedy, latte_macchiato_is_coffee, telling, cappuccino_is_coffee, eating_with_sticks, eating_with_fork, Buddhism, never_put_mayonnaise_on_pizza
$T_{Pedro}(s_1)$	Dante_Alighieri_wrote_Divine_Comedy, latte_macchiato_is_coffee, cappuccino_is_coffee, eating_with_fork, Christianity
$T_{Toru}(s_1)$	Meiji_era_was_in_1868_1912, cappuccino_is_coffee, eating_with_sticks, Buddhism, memorizing
$T_{Maria}(s_1)$	Dante_Alighieri_wrote_Divine_Comedy, latte_macchiato_is_coffee, cappuccino_is_coffee, eating_with_sticks, eating_with_fork, Christianity
$T_{Andrea}(s_1)$	Dante_Alighieri_wrote_Divine_Comedy, latte_macchiato_is_coffee, cappuccino_is_coffee, eating_with_fork, Christianity

Table 2. Traits of agents in the example in state s_1.

Note that we do not introduce types of traits and use them in the example only for convenience. One might propose a different classification of traits, e.g. putting *eating_with_sticks* as a norm. We believe that there is no single classification and it is better to deal with generic traits rather than with specific types of cultural content.

We distinguish behavior as a particular kind of traits and assume that performing a behavior by an agent changes the state of the world.

In line with AI literature, we define *behaviors* as ``[...] reified pieces of activity in which an agent engages, for example ‘sleep’ or ‘eat.’ In colloquial English an agent behaves in various ways; in technical AIese, an agent has various behaviors" (Sengers, 1998).

We define the set of all behaviors $B \subseteq T$ and the function *perform* in $Ag \times B \times S \rightarrow S$. The intended meaning of this function is that an agent, which has some behavior in some state, performs this behavior in this state and the state of the world changes to another state. More specifically, $s_v = perform(a, \tau, s_u)$ means that $has(a, \tau, s_u)$ and the agent a performed a behavior τ in the state s_u and the resulting state is s_v. The fact that $has(a, \tau, s_u)$ does not imply that the agent a is able to perform the behavior τ in the state s_u, because some preconditions for performing the behavior may be not fulfilled in the state s_u.

Note that since traits are not innate, by assuming $B \subseteq$ T we do not include innate behaviors, such as blinking when air is puffed in someone's eye.

We assume that the states are ordered, we define recursively the order ``is before" and the corresponding predicate $is\_before(s_u, s_v)$ and $is\_after(s_v, s_u)$ in the following way:

Definition 1. ***Is_before.*** $is\_before(s_u, s_v) \leftrightarrow \exists a \in Ag, \tau \in B, s \in S$ *such that* $s = perform(a, \tau, s_u) \wedge (s = s_v \vee is\_before(s, s_v))$.

Definition 2. ***Is_after.*** *$is\_after(s_v, s_u) \leftrightarrow is\_before(s_u, s_v)$.*

We assume that in each state s_v, the previous state s_u is uniquely defined, while the next state depends on the action an agent performs in s_v. We also state the following axiom:

Axiom 1. For all agents $a \in Ag$, for all behaviors $\tau \in B$ and for all states $s_u, s_v \in S$

$$s_v = perform(a, \tau, s_u) \rightarrow is\_before(s_u, s_v)$$

Definition 3. ***Sharing.*** *For each pair of agents $a_i, a_j \in Ag$, for each trait $\tau \in T$, and for each state $s \in S$, a_i and a_j share the trait τ in the state s iff they both have such a trait in s:*

$$has(a_i, \tau, s) \wedge has(a_j, \tau, s) \leftrightarrow sharing(a_i, a_j, \tau, s).$$

We also assume that agents do not lose traits when the state of the world changes, as the following axiom says:

Axiom 2. For all agents $a \in Ag$, traits $\tau \in T$ and states $s \in S$:

$$has(a, \tau, s) \rightarrow \forall s_v : is\_after(s_v, s)\ has(a, \tau, s_v).$$

Example. We can write *sharing(Toru, Maria, eating_with_sticks, s_1), or sharing(Pedro, Andrea, cappuccino_is_coffee, s_1)*, etc. To avoid giving the complete list of tuples for which sharing holds, we represent them as a graph where nodes are agents and labels on each edge denote traits that are shared by the pair of agents connected by the edge, see *Figure 1* for state s_1.

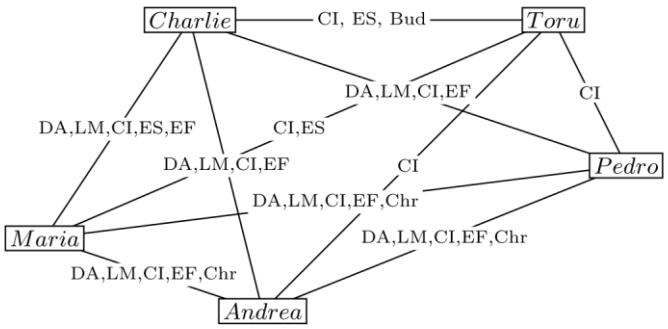


Figure 1. The graph showing for which agents and traits the predicate sharing *holds in state* s_1 *in the example. The nodes are agents and labels on each edge denote traits that are shared by the pair of agents connected by the edge. For instance, the edge between* Toru *and* Andrea *labeled* CI *means that* sharing(Andrea, Toru, cappuccino_is_coffee,s_1). *The traits are abbreviated as in Table 1: Dante_Alighieri_wrote_Divine_Comedy is abbreviated as DA, latte_macchiato_is_coffee as LM, cappuccino_is_coffee as CI, eating_with_sticks as ES, eating_with_fork as EF, Christianity as Chr, Buddhism as Bud.*

Let us assume that if an agent a_i has a trait τ, the trait τ can be transmitted to another agent a_j before some state s and we use the predicate *transmitted(a_i, a_j, τ, s)* to represent this. We represent *transmitted(a_i, a_j, τ, s)* in a graph by a directed edge from a_i to a_j labeled τ (see Figure 2).

Definition 4. ***Transmitted.*** *For each pair of agents* $a_i, a_j \in Ag$, $a_i \neq a_j$, *for each trait* $\tau \in T$, *and for each state* $s \in S$ *we say that the trait* τ *has been transmitted from* a_i to a_j *before the state* s *iff exists some state* $s_u \in S$ *such that* a_i *has* τ *in the state* s_u, a_j *does not have* τ *in the state* s_u *and an agent* a_k *performing a behavior* τ_m *in the state* s_u *imply that in the resulting state* s_v *the agent* a_j *has* τ*:*

$$(\exists s_u \in S, is\_before(s_u, s)\ has(a_i, \tau, s_u) \wedge \neg has(a_j, \tau, s_u) \wedge (s_v = perform(a_k, \tau_m, s_u)) \rightarrow has(a_j, \tau, s_v)) \leftrightarrow transmitted(a_i, a_j, \tau, s)$$

We should note that the trait τ is not shared by a_i and a_j in the state s_u, while it is shared by a_i and a_j in the state s_v, and in the state s.

From our assumption that traits are not innate, it follows that traits are acquired by agents, and the goal of the *transmitted* predicate is to show the way an agent acquired a trait. For the sake of the expressivity of the model, we assume that in the initial state agents have some traits and the way they acquire other traits is represented using the transmitted predicate.

Example. *Figure 2* shows the graph representing the *transmitted* predicate in state s_1 in our example. The traits *Dante_Alighieri_wrote_Divine_Comedy* and *eating_with_sticks* have been transmitted. On the contrary, the traits *cappuccino_is_coffee* and *never_put_mayonnaise_on_pizza* have not been transmitted (the latter trait is not even shared by any pair of agents). In particular, the *Dante_Alighieri_wrote_Divine_Comedy* trait has been transmitted from *Charlie* to *Maria*, and from *Maria* to *Andrea*. Also, the *eating_with_sticks* trait has been transmitted from *Charlie* to *Toru* and from *Toru* to *Maria*. We can write *transmitted(Charlie, Maria, Dante_Alighieri_wrote_Divine_Comedy, s_1)*.

Let us assume that in the state s_1 *Charlie* tells *Toru* that Dante Alighieri wrote the Divine Comedy. In the next state, s_2, *Toru* memorizes this piece of knowledge. This corresponds to s_2=*perform(Charlie, telling, s_1)* and s_3=*perform(Toru, memorizing, s_2)*. The *transmitted* predicate in the state s_2 is as depicted in the left part of *Figure 2* and *transmitted* in the state s_3 is as depicted in the right part of the figure. The difference in the *transmitted* predicates in these two states is that the *Dante_Alighieri_wrote_Divine_Comedy* trait has been transmitted from *Charlie* to *Toru* and the corresponding edge is added, namely *transmitted(Charlie, Toru, Dante_Alighieri_wrote_Divine_Comedy, s_3)*.

Let us also assume that in the state s_2 the set of traits for each agent is the same as in the state s_1, while in the state s_3 the following change occurs (emphasized with the bold font):

$T_{Toru}(s_3)$={*Meiji_era_was_in_1868_1912, cappuccino_is_coffee, eating_with_sticks, Buddhism, memorizing,* ***Dante_Alighieri_wrote_Divine_Comedy***}.

Obviously, the transmission has an impact on sharing and the *sharing* predicate in the state s_3 is as depicted in *Figure 3*, with the edges between *Toru* and *Charlie*, *Maria*, *Andrea*, *Pedro* added.

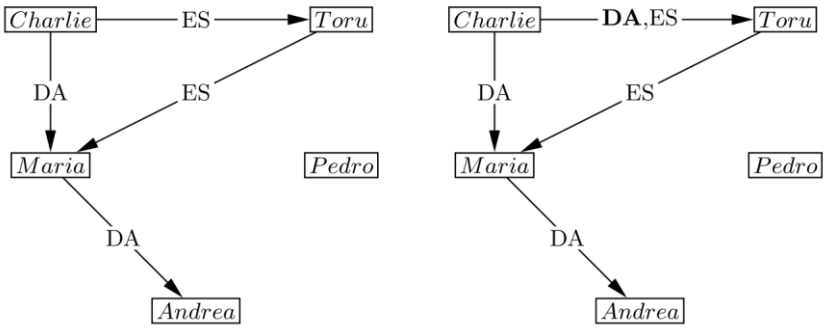


Figure 2. The graph that shows for which agents the transmitted *predicate holds in the state* s_1 *(left) and* s_3 *(right) in the example. Changes with respect to the state* s_1 *are in bold.*

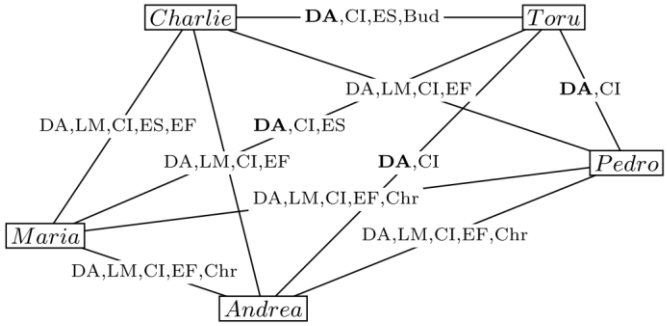


Figure 3. The graph that shows for which agents the sharing *predicate holds in the state* s_3 *in the example. Changes with respect to the state* s_1 *are in bold.*

Given a set of agents $G \subseteq Ag$ and a set of traits $T_G \subseteq T$ we define the notions of *weak sharing* and *strong sharing*.

Definition 5. ***Weak sharing.*** *A set of traits* T_G *is weakly shared by a set of agents* G *in a state* s *iff for each trait* $\tau \in T_G$ *there exists a pair of agents* $a_i, a_j \in G$, $a_i \neq a_j$ *that share* τ *in the state* s.

Definition 6. ***Strong sharing.*** *A set of traits* T_G *is strongly shared by a set of agents* G *in a state* s *iff each trait* $\tau \in T_G$ *is shared by all pairs of agents* $a_i, a_j \in G$ *in* s.

In other words, the set of traits is weakly (strongly) shared in a state *s* if it is a subset of the union (intersection) of traits shared by pairs of agents of *G* in the state *s*.

Example. Let us consider the set of agents *G={Charlie, Toru, Maria, Andrea, Pedro}*. Analyzing the *sharing* predicate in the state s_1 (*Figure 1)* we can see that only the cappuccino_is_coffee trait is shared by each pair of agents in the state s_1, so T_G*={cappuccino_is_coffee}* is strongly shared by *G* in the state s_1.

There are three traits that are shared by at least one pair of agents in the state s_1:
cappuccino_is_coffee, eating_with_sticks shared, for instance, by *Toru* and *Charlie*, and *Dante_Alighieri_wrote_Divine_Comedy* shared, for instance, by *Charlie* and *Andrea*.
So, the set $T_{G'}$*={Dante_Alighieri_wrote_Divine_Comedy, cappuccino_is_coffee, eating_with_sticks}* and all non-empty subsets of this set are weakly shared by the set *G* in the state s_1.

It is easy to see that strong sharing implies weak sharing.

Given a set of agents $G \subseteq Ag$ such that $|G| \geq 2$, and a *transmitted* predicate we introduce the notion of culture of *G*.

Definition 7. ***Weak culture of a set of agents.*** *A non-empty set of traits* $T_G \subseteq T$ *is a weak culture of* G *in a state* s *iff*

- the set T_G is weakly shared by *G* in the state *s*,
- for each agent $a \in G$ in the state s there exists a trait $\tau \in T_G$ such that has(a, τ, s).

From the assumption that traits are not innate, as we discussed, it follows that traits are acquired by agents, as represented by the transmitted predicate. Therefore, we can formulate the following axiom, telling that all traits in culture are transmitted.

Axiom 3. For each trait $\tau \in T_G$ *there exists an agent* $a \in Ag$*, that transmitted* τ *to another agent* $a_j \in G$ *before the state s, i.e.*

$$transmitted(a, a_j, \tau, s)$$

From Definition 7 and Axiom 3 it follows that all the traits in the culture are transmitted, shared, and each agent has at least one trait from the culture. Please, note that since the traits are transmitted not necessarily within the set, the transmitted predicate does not imply sharing between the agents of *G*.

Definition 8. ***Strong culture of a set of agents.*** *If* T_G *in Definition 7 is also strongly shared in the state* s *then it is a strong culture of the set of agents* G *in the state* s.

In the following if we refer to ``a culture of a set of agents", we mean ``a weak culture of a set of agents".

Example. Considering *G={Toru, Andrea}* in the state s_3, T_G*={Dante_Alighieri_wrote_Divine_Comedy, cappuccino_is_coffee}* is strongly shared by the set *G* in the state s_3.

Even though the *Dante_Alighieri_wrote_Divine_Comedy* trait has been transmitted both to *Toru* and *Andrea* from outside (from *Charlie* and *Maria*, respectively), it is strongly shared by the agents of *G*. Since in the state s_3 each agent in *G* has the trait *Dante_Alighieri_wrote_Divine_Comedy*, $T_{G'}$*={Dante_Alighieri_wrote_Divine_Comedy}* is a culture of *G* in the state s_3. It is easy to see that $T_{G'}$ is not a culture of *G* in the states s_1 and s_2 because *Toru* does not have this trait in those states.

To deal with states at a more abstract level we introduce the notion of scene as an architectural abstraction of a state.

Definition 9. ***Scene.*** *A scene* c *is a pair* c.O *and* c.Ac*, where* $c.O=\{o_i\}$ *is a set of objects and* $c.Ac=\{\alpha_j\}$ *is a set of actions.*

We denote the set of all scenes as *C*.

The notion of scene is used to represent a context, i.e. the subset of the environment faced by an agent in a state right before performing an action. Objects in the scene are treated as objects on which the agent can perform actions, and the actions in the scene are treated as actions that can be performed on these objects. In the following, we will call such actions, *possible actions*.

Let us consider a scene *c*, $c.Ac=(\alpha_1, \ldots, \alpha_k)$, $c.O=(o_1, \ldots, o_l)$ that contains *k* possible actions and *l* objects. We introduce the notion of *probability of performing an action α in the scene c*, denoted as $p(\alpha|c)$. We can use observations about past actions of a set of agents to estimate such probabilities.

Definition 10. ***Expected action.*** *An action* α *is an expected action in a scene* c *iff*

$$p(\alpha|c) = \max_{\alpha_i \in c} p(\alpha_i|c).$$

Note that there can be more than one expected action in a scene.

Definition 11. ***Cultural theory.*** *A cultural theory, denoted as* θ*, is expressed by a set of rules of the form:*

$$A_1 \wedge \ldots \wedge A_n \rightarrow C_1 \wedge \ldots \wedge C_m.$$

Here $A_1 \wedge \ldots \wedge A_n$ is the *antecedent* and $C_1 \wedge \ldots \wedge C_m$ is the *consequent*. Each element of the antecedent and of the consequent is either an action *α*, or a temporal predicate that represents a time constraint.

The rules of the theory should be interpreted as *if...then* rules that express the idea that ``if in the past the antecedent has happened, then there the consequent will happen''. We describe the rules of cultural theory in detail and we show that SICS architecture transfers behaviors in the form of such rules in the next section.

Definition 12. ***Implicit culture relation.*** *A set of agents* G' *in a state* s' *is in the implicit culture relation with a set of agents* G *in a state* s *for a set of traits* T *iff*

- is_after(s',s)
- *T* is a culture of *G* in the state *s*,
- *T* is a culture of *G'* in the state *s'*,
- *T* is not a culture of *G'* in the state *s*,
- agents of *G'* do not perform explicit actions to acquire traits from *T*.

By the last item in this definition we mean that traits from *T* are acquired implicitly, without, for instance, enumerating all traits from *T* and the current culture of G' and intending to acquire those which are not yet in the culture. Another justification of the word ``implicit'' in the name of the relation is that the definition does not refer to the internal states of the agents, i.e. to their beliefs, desires, or intentions, and, in general, to any knowledge about the set *T* or the composition of *G* and *G'*.

In the next section, we describe the Implicit Culture Framework that aims at achieving the implicit culture relation between G and G' for traits that are behaviors represented as a cultural theory. Behaviors in this context refer to the behaviors of following action patterns in the cultural theory, i.e. the fact that the consequent follows the antecedent.

Discussion

In this section we compare our definition of culture with existing definitions and discuss some properties and limitations of our approach. A more extensive analysis of the related work is provided in (Birukou et al., 2009)

Carley (1990) considers culture as the distribution of information (ideas, beliefs, concepts, symbols, technical knowledge, etc.) across the population and proposes a model for knowledge transfer based on interactions. In that model, the probability of an interaction between two agents is based on the principle of homophily, i.e. the greater the amount of knowledge they share the more probable the interaction is. During an interaction, agents exchange facts, so after the interaction one of the agents might know more than before the interaction. The knowledge transfer in these settings can be seen as a particular kind of culture spread. With respect to the definition of culture we propose in this paper, that model of information diffusion is complementary, because it models transmission of elements of culture (e.g., beliefs, knowledge) in a society.

Axelrod (1997) considers culture as a list of features or dimensions. Each feature represents an individual attribute that is subject to social influence and can have different values called traits. Two individuals have the same culture if they have the same traits for all features. Similarly to the work by

Carley, feature of an agent can change its value during an interaction and the probability of interaction is based on the homophily.

The notion of trait we use in our formalism is similar to the notion of feature used by Axelrod, specifically, each feature can take value from a set of specific traits.

Traits in our formalism also includes ideas, beliefs and technical knowledge used as culture elements by Carley. Both theories by Carley and by Axelrod are based on the assumption that culture changes as a result of an interaction. Thus, in our terms, interaction in that sense can be considered as a particular kind of transmission: there are two agents participating, it takes place in some specific state and it leads to the appearance of some cultural element in one of the agents.

Hofstede (2001) treats culture as ``[...] the collective programming of the mind that distinguishes the members of one group or category of people from another", proposes a model of culture and applies it for studying and comparing cultures of IBM workers in more than 50 countries. The model includes the following five independent dimensions of national culture differences: *power distance*, which is related to the different solutions to the basic problem of human inequality; *uncertainty avoidance*, which is related to the level of stress in a society in the face of an unknown future; *individualism* versus *collectivism*, which is related to the integration of individuals into primary groups; *masculinity* versus *femininity*, which is related to the division of emotional roles between men and women; and *long-term* versus *short-term* orientation, which is related to the choice of focus for people's efforts: the future or the present. Values in Hofstede's terms refer to ``a broad tendency to prefer certain states over others" and are similar to attitudes and beliefs, which are just particular kind of traits in our formalism. Dimensions, similarly to Axelrod's features, take values from the set of traits. Thus, comparing with our work, the model developed by Hofstede has a different focus - it aims at comparing cultures of groups of people over several pre-defined dimensions of values, while our model supports comparison over arbitrary sets of traits. The dimensions in Hofstede's model are meant to be independent, while our formalism does not address the issue of dependency of traits, so they can be dependent on each other. In this line of thoughts, an interesting application of our model could be comparison of dependency of traits across groups, i.e. if presence of a trait or traits leads to the presence of another trait(s) for one group and to the presence of third trait(s) for another group.

The definition of culture presented here allows for representation and comparison of different cultures. However, in order to compare traits, one first needs to identify the traits of individuals. On the one hand, deducing traits from manifested behaviors of agents is not a trivial task in general. On the other hand, in specific domains this might be much easier, consider, for instance, deducing traits of users from logs of a web service, website, or an application. For instance, it would be possible to see that a group of users of a text editor always turn off the autocorrect feature and turn it off automatically in new versions of the editor prepared for this group. Taking the issue of the observability of traits into account, we see social software and Web 2.0 systems as one of the potential application domains for our model.

Space restrictions do not allow us to present all features of our model. However, we would like to emphasize that our model (see, e.g. (Birukou, 2009), (Birukou et al., 2009)), and a working paper (Birukou et al., 2009a) supports evolution, thus agents can acquire new traits and the model captures the change of culture and allows for comparison between states.

Also, as a motivation for inclusion of the “implicit” aspect in the model, let us recollect that traits rarely exist in isolation, rather, they are related to each other, and, depending on the individual, the transmission of one trait may lead to appearance of other traits. For example, let us imagine that *Michael* tells *Li* that *Company* released a new web browser, *Browser*. Even though *Li* never saw *Browser*, she can guess that using *Browser* it is possible to visit web pages, play videos online, and so on. So, the transmission of a single piece of knowledge *Browser_is_browser* lead to appearance of such behavior as *Visit_homepage_using_Browser*, *Watch_videos_using_Browser*, etc. Therefore, we argue that in practice, transferring some traits from G to G' may result in transferring a bigger set of traits. An example further supporting our argument can be found in (Kuroda and Suzuki, 1991, pp.

26,30), where the authors observed that by learning English, Arab students learned ``something else'', namely some implicit elements of Western culture.

Defining an appropriate set of traits for the considered domain is not an easy task. The modeler should select the right level of granularity to avoid extreme cases of having just several traits and too many traits. Just to give an example, let us discuss the distinction between *action* and *behavior*. In AI literature, an action is an atomic piece of activity, while behavior is perceived as something more complex, and can include several actions. Therefore, our notion of performing a behavior can really be decomposed into performing several actions. However, we decided not to introduce explicit relations between actions and behaviors. Moreover, the absence of such clear dependency in AI literature suggests that these relations are hard or even impossible to formalize. Instead, we assume that behavior can represent an atomic action or a more complex activity depending on the level of modeling granularity. We can vary granularity of behaviors depending on the problem in hand and on the domain. For instance, in the running example, when someone needs to know whether agents are working, it is possible to consider behaviors *working* and *playing*, or, even, *working* and *not_working*. However, if someone would like to have a closer look at leisure activities of the group, it is necessary to introduce finer granularity of the *playing* behavior, e.g. by considering *playing_basketball* and *playing_chess* behaviors.

THE IMPLICIT CULTURE FRAMEWORK

In this section we present the meta-model of implicit culture concepts and then describe the architecture of a System for Implicit Culture Support (SICS) that achieves implicit culture relation between two sets of agents.

Meta-model

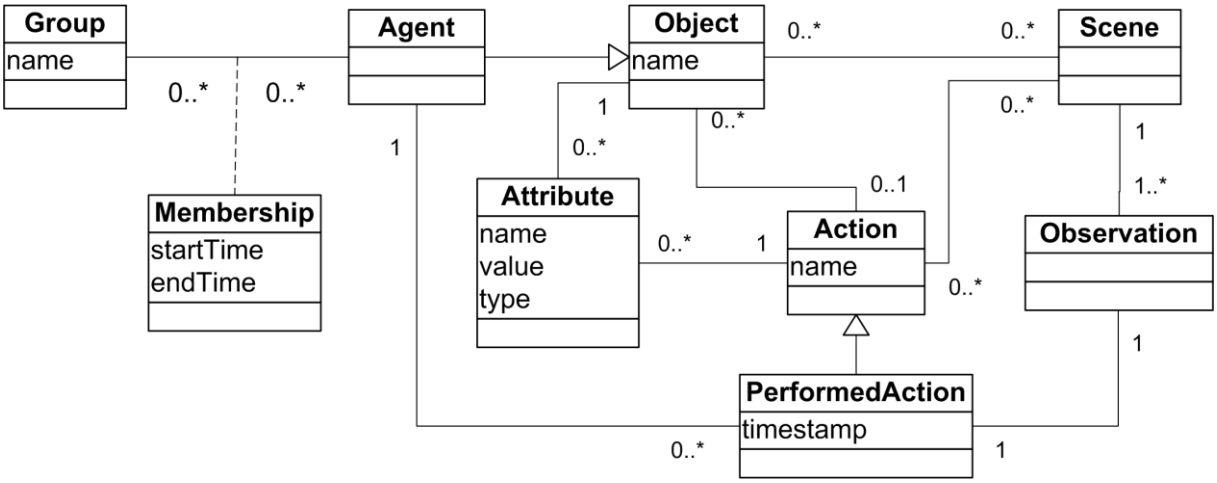


Figure 4. The meta-model of the Implicit Culture concepts.

The meta-model that illustrates relations between the core Implicit Culture concepts is shown in Figure 4. The environment is described in terms of *agents* that perform *actions* on *objects*. An *object* is defined by its name and a set of related attributes. *Attributes* represent additional information about objects, actions, or agents and consist of a name, a value, and the type of the value. An *agent* is a particular type of object that can perform actions. Several agents can be referred to as a *group*. An agent's *membership* in the group can be restricted in time. An *action* is characterized by its name, a set of related attributes, and a set of related objects. Each *performed action* is a specific kind of action that contains the *timestamp* and the agent of the action. The actions are considered in the context of scenes, where each *scene* contains the set of actions that are possible to perform, and the set of objects agents can operate with. After the agent performs one of the possible actions, the performed action and the scene constitute an *observation*.

A *performed action* is represented using the following syntax:
action_name(agent_name(ag_attribute_name1=ag_attribute_value1,...);
object_name1(o_attribute_name1 = o_attribute_value1,…),…; attribute_name1 = attribute_value1,…;
timestamp),

Thus, we start from the name of the action and then list the agent, objects, attributes, and the timestamp of the action, recursively listing attributes for the agent and objects.

An action, object, agent, timestamp, or attribute value can be a variable denoted as wildcard (*) or as a small Latin letter. For the complete syntax of the language we use to represent actions please refer to Appendix A of (Birukou, 2009).

Definition 13. ***Cultural action.*** *An action* α *is a cultural action with respect to a cultural theory* θ *iff it matches one of the atoms* C_i *of the consequent of rules of* θ.

Note that we require matching rather than equality because we assume that both cultural action and atoms of the consequent can contain variables.

General architecture of a SICS

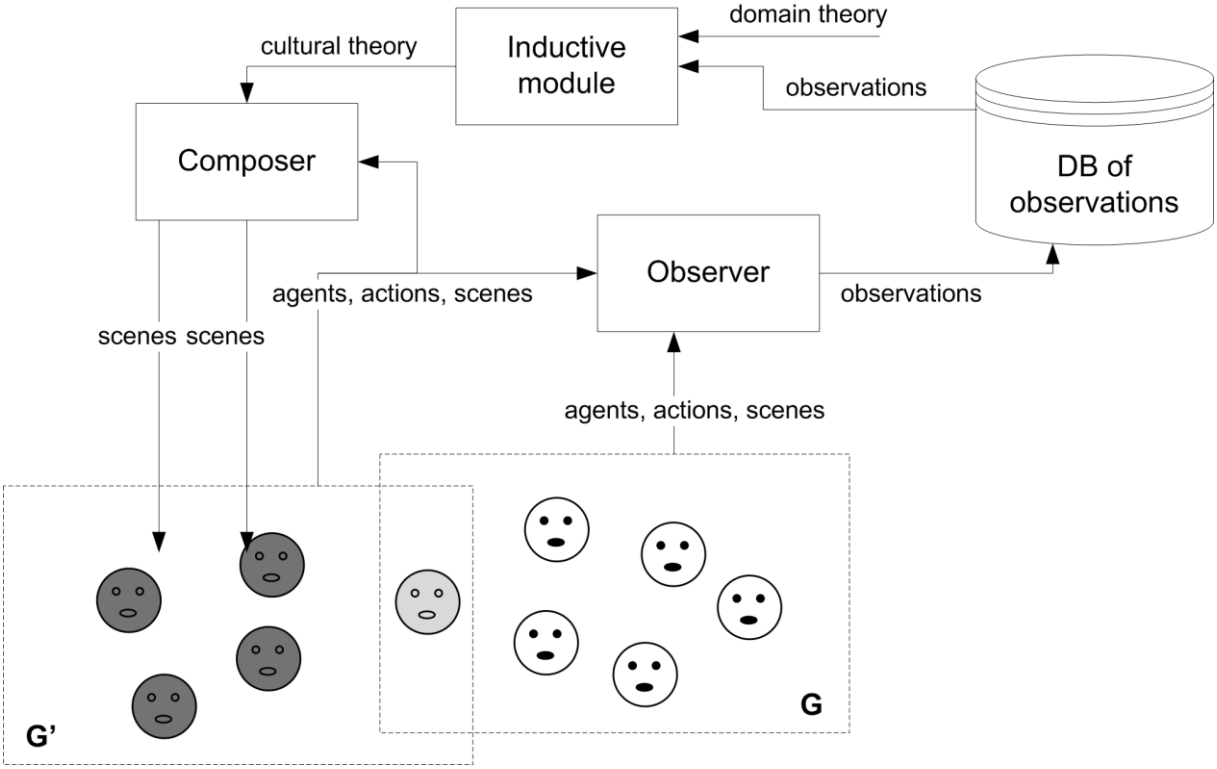


Figure 5. The architecture of a SICS.

The general architecture of a SICS is shown in meta-model that illustrates relations between the core Implicit Culture concepts is shown in Figure 5 and consists of the following three components:

- The *observer*, which collects information about actions performed by agents of *G* and *G'* in different scenes and stores this information in a database of observations;
- The *inductive module*, which analyzes stored observations of agents of *G* and applies learning techniques to find patterns of user behavior, i.e. the culture of the community represented as a *cultural theory*;
- The *composer*, which uses the information collected by the observer and the theory produced by the inductive module in order to manipulate scenes faced by the agents of *G'* in such a way that actions of *G'* are consistent with the cultural theory.

The goal of a SICS is to establish the implicit culture relation between sets of agents *G* and *G'* for a set of behaviors represented as rules of a cultural theory θ. By establishing the implicit culture relation

between *G* and *G'*, the SICS transfers culture, as represented by the cultural theory θ, from the set of agents *G* to the set of agents *G'*. In other words, the SICS tries to make agents in *G'* behave as agents in *G* behave.

The architecture achieves the implicit culture relation in the following two steps:

Step 1: Expressing *T*, a set of traits to be transferred from *G* to *G'*, as a cultural theory θ.

Step 2: Manipulating the scenes faced by *G'* in such a way that some of expected actions of *G'* in the resulting scenes satisfy θ.

Both steps are performed using observations about actions of agents of *G* and *G'*. It is important to note that in practice, the cultural theory to be transferred is not pre-defined, but must be discovered. The proposed SICS architecture addresses this problem by means of the inductive module. In a general case, we assume that the cultural theory θ consists of two parts. The first part, θ_0, called *domain theory* consists of the pre-defined rules of behavior to be transferred from *G* to *G'*. The second part is learned by inductive module.

The first step of achieving the implicit culture relation leads to the problem of induction of the cultural theory. Let us re-formulate this problem as follows:

Inductive Module Problem. Given a set of performed actions of the agents of *G*, find a cultural theory θ about their actions.

The inductive module problem is a rather standard learning problem: inducing the patterns of behavior of a group given a set of observation. This problem can be solved using standard data mining techniques given a proper choice of the language for expressing the cultural theory.

Now let us describe the composer module of the general SICS architecture in detail and present algorithms used in the composer.

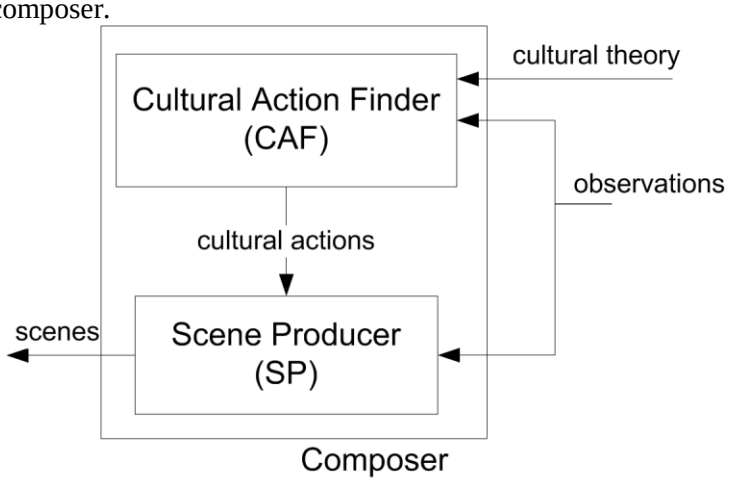


Figure 6. The architecture of the composer module.

Recollecting how a SICS achieves the implicit culture relation, the goal of the composer is to propose a set of scenes to agents of *G* such that the expected actions of these agents in these scenes satisfy the cultural theory θ. In our implementation, the composer consists of two main submodules, as shown in Figure 6:

- The *Cultural Action Finder (CAF)*, which takes as inputs the theory θ and the observations of *G'*, and for the most recent observation that matches the antecedent of one of theory rules, CAF produces as output the cultural actions, i.e. the actions from the consequents of the fired rule of θ.
- The *Scene Producer (SP)*, which takes the cultural actions produced by the CAF and, using the observations of *G* and *G'*, for each cultural action produces a scene such that the cultural action is among expected actions in the scene.

As we mentioned earlier, there can be more than one expected action in a scene. Therefore, we require that the cultural action is among expected actions in the scene. A possible implementation can give priority to scenes where the cultural action is the only expected action, and if there are no such

scenes, find a scene where the cultural action is one of expected actions. Note also that in general, CAF might return several cultural actions. In such a case, SP finds a scene for each of the cultural actions and returns a set of scenes.

Thus, the second step of achieving the implicit culture relation leads to the problem of prediction of scenes. Let us formulate this problem as:

Scene Producer Problem. Given a set of performed actions of the agents of G and G', and given a cultural action α for an agent $a \in G'$, find a scene c such that α is among the expected actions of a in the scene c.

The most important aspect of the scene producer problem is the requirement of the effectiveness of the scene with respect to the *persuasiveness* of the scene, i.e. how the scene helps to achieve the goal of having a specific action performed. The scene producer problem is different from classical supervised or unsupervised classification problems and clustering.

In the following subsections we describe the details of the algorithms implemented by the two modules.

Cultural Action Finder

The CAF matches the observations of G' with the antecedents of the rules of θ. The CAF starts with the most recent observation, then moves to the second last if the most recent observation does not match any rule, and so on. If the CAF finds an observation that matches the antecedent of a rule, then it takes the consequent of the rule as a cultural action. Figure 7 presents the algorithm of the CAF. For each rule **r** (**ant→cons**), the function *match*(ρ, α) checks whether the atom ρ of **ant**=*ant* (**r**) matches the action α; then the function *find-set*(**ant, past-actions**) finds a set of **past-actions** of past actions that match the set of atoms of **ant**; and finally, the function *join*(**past-actions, r**) joins the variables of **r** with the actions in **past-actions**, i.e. it fills the corresponding variables in the rule **r** with values from **past-actions**. The function *cons*(**r'**) returns the consequent of the resulting rule **r'**.

```
loop
  get the last performed action α
  for all rule r of θ do
    for all atom ρ of ant(r) do
      if match(ρ,α) then
        if find-set(ant,past-actions) then
          r′=join(past-actions,r)
          return cons(r′)
        end if
      end if
    end for
  end for
  return null
end loop
```

Figure 7. The algorithm of the CAF submodule.

Scene Producer

For each of the cultural actions found by the CAF, the SP tries to find a scene where the cultural action is the expected action. Thus, given a cultural action α for the agent $a_1 \in G'$ that performed actions in the set of scenes $C(a_1)$, the algorithm used in SP consists of three steps:

find a set of agents $G_0 \subseteq G \cup G'$ that performed actions similar to α and the sets of scenes $C(a)$, $a \in G_0$, in which these agents performed actions;

select a set of agents $G_0' \subseteq G_0$ most similar to a_1;

estimate (using G_0) the similarity between the expected actions of a_1 in the scenes of the set $C = \bigcup_{a \in G_0} C(a)$ and the cultural action α. Return the scene that maximizes the similarity and propose it to a_1.

Figure 8 shows the simple algorithm used in the first step in SP. An agent a is added to the set G_0 if the similarity $sim(\alpha_a, \alpha)$ between at least one of its performed actions α_a and α is greater than the minimal similarity threshold min_{sim}. The scenes $\{c\}$ in which the α_a actions have been performed are added to $C(a)$, which is the set of scenes in which a has performed actions similar to α. At this point, we do not specify how the similarity between actions is calculated. We just assume that it is a function that can be either generic or domain-specific, and its values range from 0 (not similar at all) to 1 (the same). One can find examples of such function in (Birukou et al., 2007a).

```
for all a ∈ G ∪ G′ do
  for all performed actions α_a of a do
    if sim(α_a,α)> min_sim then
      if a ∉ G_0 then
        a → G_0
      end if
      c → C(a)
    end if
  end for
end for
```

Figure 8. The algorithm for the first step in the SP submodule.

In the second step, the SP algorithm selects k neighbors in G_0 in such a way that these neighbors are most similar to a_1 with respect to the function of similarity between two agents, defined as follows:

$$sim(a_1, a) = \frac{1}{|C(a_1) \cap C(a)|} \sum_{c \in C(a_1) \cap C(a)} \frac{1}{|Ac_{a_1}(c)||Ac_a(c)|} \sum_{\alpha_{a_1} \in Ac_{a_1}(c)} \sum_{\alpha_a \in Ac_a(c)} sim(\alpha_{a_1}, \alpha_a)$$

Equation 1. Similarity between two agents as defined for the second step of the SP algorithm.

where $C(a_1) \cap C(a)$ is the set of scenes in which both a_1 and a performed at least one action. $Ac_{a1}(c)$ and $Ac_a(c)$ are the sets of actions that a_1 and a, respectively, have performed in the scene c. Essentially, this similarity function defines the similarity between two agents as the similarity between their actions in scenes where they both performed actions. Equation 1 can be replaced with a domain-dependent agent similarity function, if needed.

In the third step, the SP algorithm selects the scenes in which the cultural action is the expected action. To do this, we first estimate the similarity value between the expected action of a_1 and the cultural action for each scene $c \in C = \bigcup_{a \in G_0} C(a)$, and then select the scene with the maximal value. The function to be maximized is the expected value $E(sim(\alpha_{a1}, \alpha)|c)$, where α_{a1} is the action performed by the agent a_1, α is the cultural action, and $c \in C$ is the scene in which α_{a1} is situated. The following estimate is used:

$$\hat{E}\left(sim(\alpha_{a_1}, \alpha)|c\right) = \frac{\sum_{a_i \in G_0'} E\left(sim(\alpha_{a_i}, \alpha)|c\right) \cdot sim(a_1, a_i)}{\sum_{a_i \in G_0'} sim(a_1, a_i)}$$

Equation 2. An estimate of the conditional similarity between two cultural actions.

which means we calculate the weighted average of the similarity of the expected actions of the neighbors of a_1 in the scene c, where the weight $sim(a_1,a_i)$ is the similarity between the agent a_1 and

the agent a_i, whereas $E(sim(\alpha_{ai},\alpha)|c)$ with $a_i \in G_0'$ in Equation 2, to avoid recursion, is estimated as follows:

$$\hat{E}\left(sim(\alpha_{a_i},\alpha)|c\right) = \frac{1}{|Ac_{a_i}(c)|} \sum_{\alpha_{a_i} \in Ac_{a_i}(c)} sim(\alpha_{a_i},\alpha),$$

which is the average of $sim(\alpha_{ai}, \alpha)$ over the set of actions $Ac_{ai}(c)$ performed by a_i in c.

The algorithms described above are fully implemented in Java using XML for expressing the cultural theory. However, the algorithms given here are only one possible implementation, and they can be further refined or modified. For instance, in the second step we can consider not only the similarity between agents based on their actions, but also general similarity between agents based on their names and attributes. This would correspond to the following equation:

$$sim(a_1,a) = \gamma \cdot sim(a_1,a) +$$
$$+\frac{1-\gamma}{|C(a_1)\cap C(a)|} \sum_{c\in C(a_1)\cap C(a)} \frac{1}{|Ac_{a_1}(c)||Ac_a(c)|} \sum_{\alpha_{a_1}\in Ac_{a_1}(c)} \sum_{\alpha_a \in Ac_a(c)} sim(\alpha_{a_1},\alpha_a),$$

Equation 3. An example of calculating the similarity between agents based on their names and attributes.

which is a modified Equation 1, $sim(a_1,a)$ is the similarity between agents a_1 and a, and $0 \leq \gamma \leq 1$ is a coefficient that defines which similarity (the one between agents, or the one between actions which agents performed) has more weight. Equation 1 is obtained from Equation 3 taking $\gamma=0$.

The IC-Service: an implementation of the Implicit Culture Framework

In this section, we describe the IC-Service that implements the Implicit Culture Framework. It is a multi-purpose web service which provides simple and configurable access to the SICS described in previous sections. We have chosen the web service technology among the possible solutions because it follows the Service-Oriented Architecture (SOA) paradigm supporting principles of universal access and platform independence. Applications of the Implicit Culture Framework have a direct dependence on the domain and must be customizable. Therefore, configurability and extensibility without code modification became the main focus of our work on the IC-Service.

The architecture, implementation details and invocation scenarios of the IC-Service are presented in (Birukou et al., 2007). In the following, we describe the details of the core SICS modules (Composer, Inductive and Observer) as implemented in the IC-Service (Figure 9).

The SICS Core

The SICS Core implements the detailed SICS architecture, providing the means for managing observations, the cultural theory, and recommendations. The main functionality of the **Composer Module** (Figure 9) is to provide recommendations, and it also contains *Similarity Utilities*, which implement the algorithms for calculating the similarity between objects, actions, etc., and *CAF Utilities* used by the Cultural Action Finder submodule for finding actions consistent with the theory. To discover a theory that expresses patterns in users' behavior, the **Inductive Module** (Figure 9) incorporates the implementation of the *Apriori Algorithm* for the association rule mining (Agrawal & Srikant, 1994) and its extension for generating rules in the *Apriori Rule Generator*. The dashed line shows that the functionality of the module can be extended with other learning techniques.

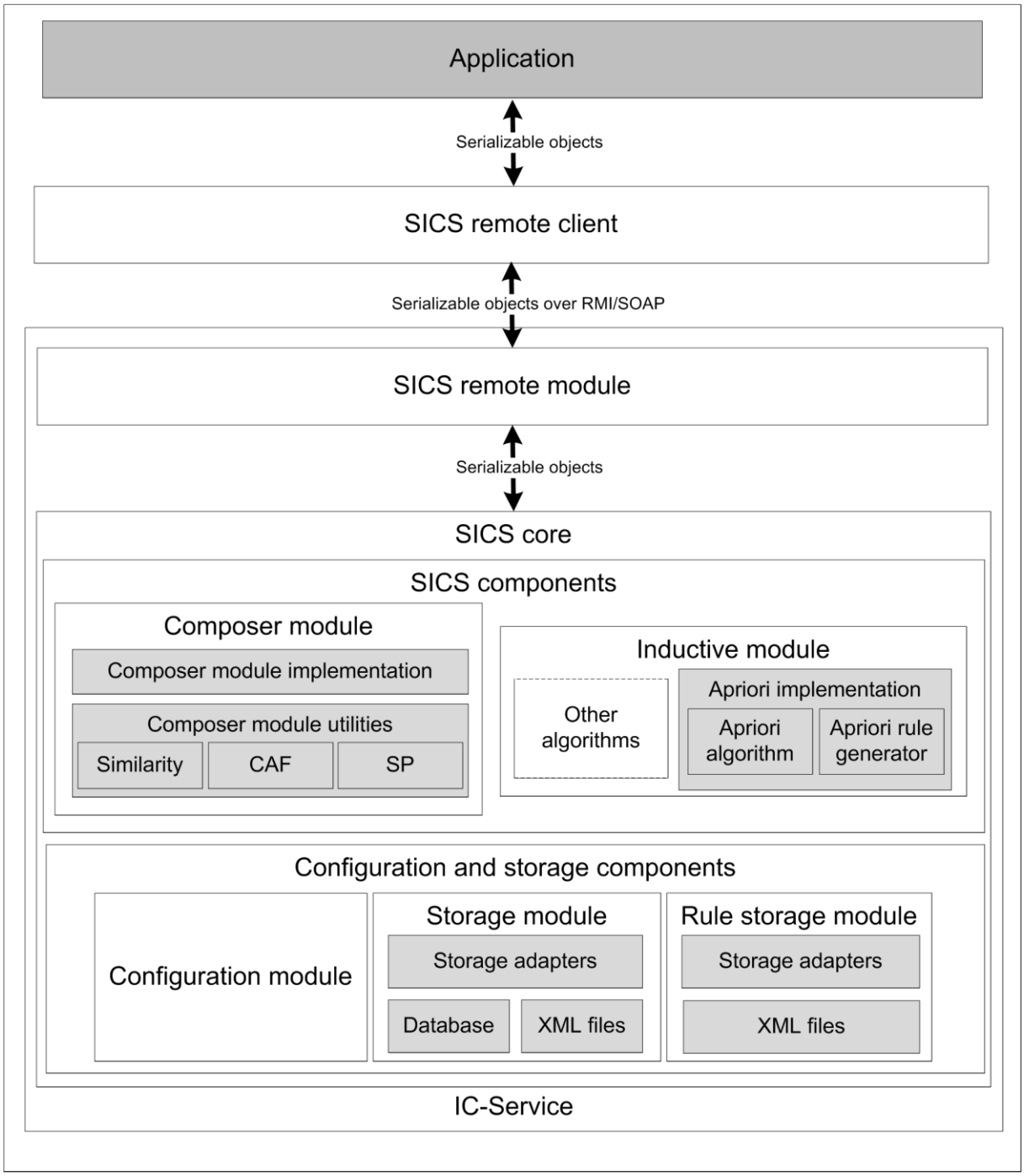


Figure 9. The architecture of the IC-Service.

All parameters of a SICS instance are configured in the **Configuration Module**. Each instance of the SICS can have different configurations of the composer module, the mechanism of processing the theory in the inductive module, and similarity algorithms. Similarity configuration is stored in XML files.

The details of the **Storage Module (Observer)** are shown also in Figure 9. This module is responsible for storing information about the application domain, i.e., it can be used to add or delete agents, manage groups, and save observations. Thus, this model implements the functionality of the *Observer Module* in the general SICS architecture. The SICS can use one of the following two modules to store data: the *Database Storage Module* stores the data in a RDBMS whereas the *XML Storage Module* stores the information in XML files.

The **Rule Storage Module** is responsible for the management of the theory. For instance, it can be used to add or remove theory rules. The internal architecture is similar to the architecture of the Observer Module, however, the Rule Storage Module supports only XML storage facilities.

The cultural theory

The IC-Service supports the adjustment of a desired behavior of a group through configuring rules of the cultural theory. The general description of a cultural theory was given in previous sections. In this section, we describe the implementation of the theory in the IC-Service. The meta-model of the cultural theory is shown in Figure 10. A rule of the theory is defined in the form
if consequent **then** antecedent,
where consequent and antecedent consist of one or several predicates. The intuition is that if consequent happened then antecedent will happen.

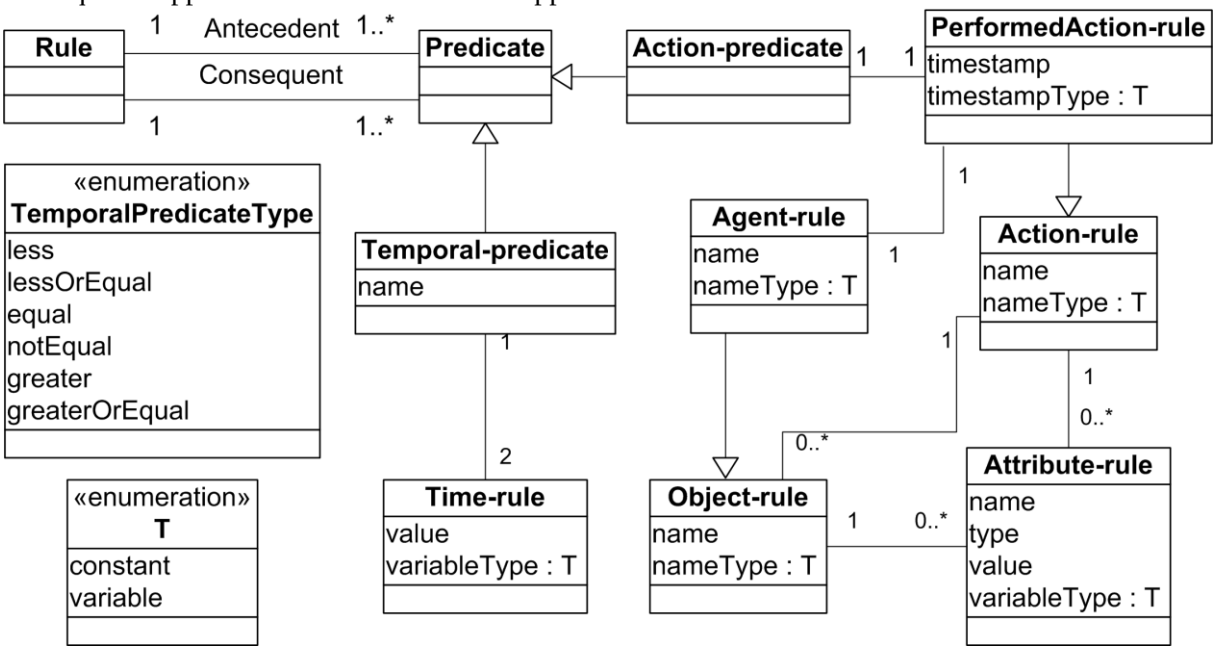


Figure 10. The meta-model of the cultural theory.

An example of the theory telling that if someone pressed ``stop" button in an Italian bus, then this person is exiting the next bus stop, can be expressed as

if *press(_a; stop_button;; t)* **then** *exit(_a; next_stop;; t+1)*

For a recommendation system, an example of the simplest recommendation strategy can be expressed as

if *request(_a;;request-params=...; t)* **then** *rate_high(_a; recommendation;request-params=...;t+1)*

which means that recommendations for each user request must then obtain high ratings.

Each predicate describes either conditions on observations (*action-predicates*) or conditions on time (*temporal-predicates*). A temporal-predicate includes a predicate name that shows the semantics of the predicate, e.g. ``less" or ``equal", and two time-rules that impose constraints on timestamps of the compared performed actions. Each action-predicate contains one *performedAction-rule*, which specifies conditions on the performed actions. A performedAction-rule may specify conditions on the agent that performed the action and also, being an action-rule, it specifies patterns on objects and attributes of the action. In all rules names and elements can be *constants* or *variables*.

```
<?xml version="1.0" encoding="UTF-8"?>
<rules xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="sics_rules.xsd">
 <rule identifier="IC4WebServices">
  <antecedents>
   <action-predicate>
    <action-rule name="request" timestamp="*" timestamp_type="variable" name_type="constant">
     <agents>
      <agent-rule name="*" name_type="variable" />
     </agents>
     <objects>
      <object-rule name="_x" name_type="variable">
       <attributes>
        <attribute-rule type="String" variable_type="variable" name="keyword">_y</attribute-rule>
       </attributes>
      </object-rule>
     </objects>
    </action-rule>
   </action-predicate>
  </antecedents>
  <consequents>
   <action-predicate>
    <action-rule name="apply" timestamp="*" timestamp_type="variable" name_type="constant">
     <agents>
      <agent-rule name="*" name_type="variable" />
     </agents>
     <objects>
      <object-rule name="_x" name_type="variable">
       <attributes>
        <attribute-rule type="String" variable_type="variable" name="keyword">_y</attribute-rule>
       </attributes>
      </object-rule>
      <object-rule name="*" name_type="variable">
       <attributes>
        <attribute-rule type="String" variable_type="variable" name="operation_name">*</attribute-rule>
       </attributes>
      </object-rule>
     </objects>
    </action-rule>
   </action-predicate>
  </consequents>
 </rule>
</rules>
```

Figure 11. An example of the XML representation of the cultural theory in the IC-Service.

For all rules, names and values can be constants or variables, depending on nameType and valueType parameters. If a name or a value is a constant, the corresponding elements are considered only if they are equal to this pre-defined constant. In case of a variable, all elements that match the defined structure are selected, regardless of their values. There are two options of specifying a variable: using a wildcard (*****), meaning that the element takes any value, and using **_someName** structure, which means that the value can be any, as long as all values (there might be several occurrences of **_someName** within the same rule or in different rules) **_someName** takes in the theory are the same.

An example of a cultural theory is given in Figure 11. It can be represented in the language we use as follows:

if *request(*;_x(keyword=_y);;*)* **then** *apply(*;_x(keyword=_y),*(operation_name=*);;*),*

and it means that **if** someone is requesting [web service] for the problem *_x* described by the attribute *keyword=_y*, **then** the returned service (specified with the attribute *operation_name=**) is applied for the problem *_x*.

The described theory rules are used by the composer module to analyze observations from the SICS storage. When an agent performs an action, the observation corresponding to the action is matched with the antecedent part of the theory. The corresponding consequents, where non-wildcard variables may be assigned corresponding values from antecedents, are called *cultural actions* and used in the algorithm for providing recommendations. The details of the algorithm used to match observed actions with the theory are provided in previous sections.

The cultural actions are used to find scenes where actions similar to cultural actions happened. The IC-Service provides a simple algorithm that calculates the similarity between pairs of actions using predefined similarity weights for names, timestamps, agents, objects, and attributes of the actions. These values can be configured for each particular type (action, object, agent, or attribute), for each particular instance of the element, or for particular pairs of elements. We do not present technical details of the similarity configuration in this chapter, but the algorithm is conceptually similar to the one described by Spanoudakis and Constantopoulos (1996).

If an application requires a custom algorithm for calculating similarity between particular kinds of elements, then it can be easily added into the system using the configuration file. For instance, in the web service discovery system, described later in this chapter, some attributes were compared using WordNet-based similarity metric, while in the IC-Patterns system (Birukou et al., 2006) an ad-hoc algorithm for calculating similarity between user queries has been used.

Applying the Implicit Culture Framework in a particular scenario: a methodology

In this section we describe how to apply the Implicit Culture Framework in a specific scenario. We provide a set of steps to be performed when applying our approach. At the moment, we do not provide any strict requirements on how the steps should be performed, leaving the choice of tools and methods open. Thus, the steps should be considered only as guidelines. However, some ideas can be obtained by looking on how we apply the Implicit Culture Framework in the system for web service discovery.

In general, for using the Implicit Culture Framework, the following steps must be accomplished:

1. Describe the application domain in the terms of the meta-model of the implicit culture concepts.
2. Define the domain theory.
3. Choose how to use the IC-Service in the application.
4. Configure the observer module, i.e. decide which actions, objects, attributes will be stored.
5. * Configure the inductive module, i.e. decide which algorithms will be used, how often they will be applied to learn a theory, and how often the learned theory will be merged with the domain theory.
6. Define algorithms for calculating the similarity between agents, actions, objects, attributes.
7. Configure the composer module, i.e. how many scenes are proposed, define similarity thresholds, who belongs to group *G* and who belongs to *G'*, etc.

The steps which are not supported in the current implementation of the framework are marked with a star.

In the application described in the next section, we explain these steps in more detail with examples.

APPLYING THE IMPLICIT CULTURE FRAMEWORK TO THE DEVELOPMENT OF RECOMMENDATION SYSTEMS

This section describes an application of the Implicit Culture Framework in the field of recommendation systems. We describe a system that addresses the problem of web service discovery.

We first provide a brief introduction in the domain, and then we describe how we applied the Implicit Culture Framework and proceed with the description of the implemented system.

Recommendation systems

A recommendation system is a ``system that produces individualized recommendations as output or has the effect of guiding the user in a personalized way to interesting or useful objects in a large space of possible options" (Burke, 2002). The main idea of most recommendation systems is to leverage information about users and items in order to produce recommendations relevant to user interests.

The information about users is collected in terms of profiles, implicit or explicit feedback from users. Time spent reading [a web page], clickthrough rate, actions used to end a search session are among examples of implicit feedback. Examples of explicit feedback include relevance judgments and ratings.

Recommendation domains include, but are not limited to, movies (MovieLens[1]), music (JUKEBOX (Tremblay-Beaumont and Aïmeur, 2005)), books (Amazon, (Linden et al., 2003)), web links (Implicit, (Birukou et al., 2005)), and hotels (TripAdvisor[2]).

The problem of providing interesting recommendations can be formalized in terms of the Implicit Culture Framework: it is necessary to transfer behavior of selecting interesting items from those who possess this behavior to other users.

Collaborative filtering (Resnick et al., 1994) is one of the most popular techniques for the development of recommendation systems. It addresses the problem of information overload, i.e. having too many items potentially interesting for a user. The main idea of collaborative filtering is that items unseen by the user, but highly rated by similar users, should be recommended. The similarity between users is calculated based on the ratings they provided in the past.

The Implicit Culture Framework is claimed to be more general than collaborative filtering (Blanzieri et al., 2001), since it filters not only ratings, but actions in general, with *rate* being only a particular kind of the action. In the terms of the Implicit Culture Framework, the cultural theory in collaborative filtering is specified a priori and is not updated over time, therefore, the inductive module is not necessary. The composer module of the SICS for collaborative filtering plays the main role in the process of producing recommendations. It selects potentially relevant items by comparing *rate* actions. In the general architecture of the SICS, actions are not restricted to *rate* actions. Moreover, the theory can evolve over time, incorporating the essence of the history of observations.

Web service discovery

Service-oriented computing and web services are gaining more and more popularity enabling the organizations to use the Web as a market for their own services and consume already existing software. On the other hand, the more services are available the more difficult it becomes to find the most appropriate service to use in a specific application. Existing approaches to web service discovery tend to address different styles of information processing, including the development of extensive service description and publication mechanisms (Martin et al., 2004), and the use of syntactic, semantic and structural reviews of web service specifications (Keller et al., 2004). Web services have a set of functional and non-functional characteristics which may be difficult to present and control. Service behavior and Quality of Service (QoS) parameters may vary over time, better services may appear and acquire popularity in certain business areas. Developers of service-based applications may want to discover web services and replace previously exploited ones for repairing or generally improving their systems. Despite the availability of various tools, the selection often relies on the information provided by someone (business partners, experts on the field, friends, etc.) who has already gained experience with a certain service.

---

[1] MovieLens. Movie recommendations. http://movielens.umn.edu/

[2] TripAdvisor. Reviews of hotels, resorts and vacations. http://www.tripadvisor.com/

To support such information exchange, the idea of applying recommendation systems for discovering and selecting web services has been recently proposed (Bova et al., 2007; Kerrigan, 2006; Manikrao & Prabhakar, 2005; Sherchan et al., 2006).

Existing recommendation-based approaches use ratings of service providers based on explicit and often subjective opinions of service clients (Sherchan et al., 2006). However, as demonstrated in (Claypool et al., 2001), people are not usually willing to actively provide feedback. Our aim in this work is to allow developers of service-based applications to benefit from experience of other developers without requesting them to spend additional effort for evaluating services. The overall approach is to connect requests for services with observations of service invocations and executions that follow such requests. Data collected during observations are the input to identify which services are considered relevant for specific requests of a particular community of clients. Additionally, data about service execution can be used for ranking services according to their QoS. The effort requested from developers is only to enable observations of web service invocations performed by their applications. In exchange for this, such developers can benefit from accessing the history of service executions and obtain recommendations which services are better to use for their tasks.

In this section, we present an implemented system for improving web service discovery (Birukou et al., 2007a). The system is based on the IC-Service. It enables web service monitoring and recommends services based on data provided by service clients rather than information advertised by service owners. The approach can be extended to support personalized requests and learn which services can better satisfy them. Methods for matching client requests with the requests from the system history are a crucial aspect of the system. We tested two similarity metrics: (i) the classical Vector-Space Model (VSM) and (ii) a semantic matching metric that uses the WordNet[3] lexicon.

Applying the Implicit Culture Framework

In this section, we illustrate the use of the IC-Service for supporting web service discovery. The IC-Service manages the history of requests for web services, collects reports about service invocations by heterogeneous clients and helps developers to discover and select web services suitable for their applications. See Figure 12 for an overview of the overall architecture, including the role of the IC-Service. To join a community that shares experience about service usage, a developer must include into his/her application the part of the IC-Service, called SICS Remote Client, that enables monitoring of web service invocations on the client side.

[3] http://wordnet.princeton.edu/

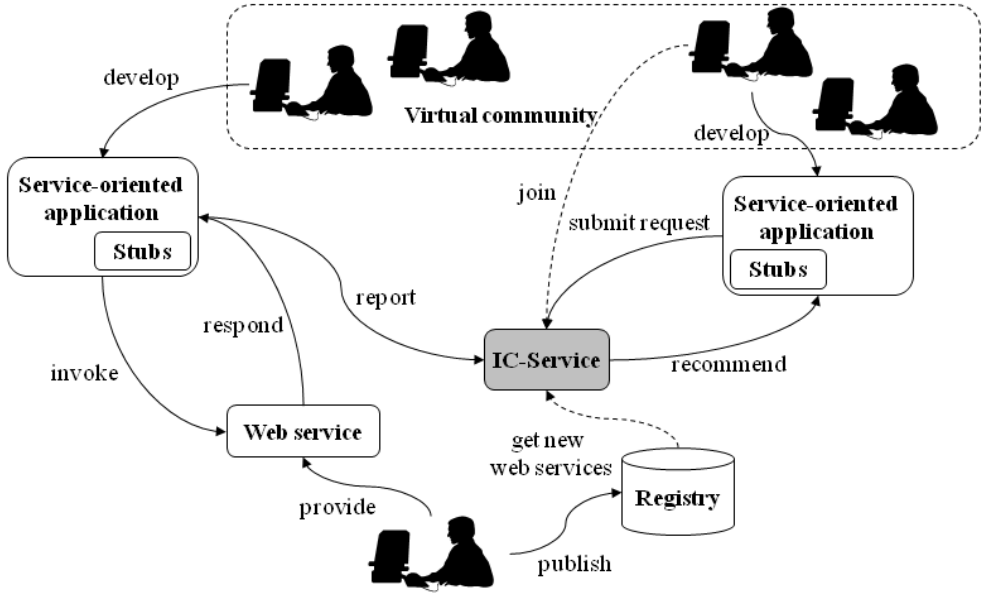


Figure 12. Web service discovery with the IC-Service.

In general, for creating a recommendation system for web services with the IC-Service, the following steps have been accomplished:

1. Formalization of the application domain in the Implicit Culture terms.
2. Definition of the cultural theory.
3. Definition of similarity calculation algorithms.

These steps correspond to steps 1, 2, and 6 from the steps of the methodology for applying the Implicit Culture Framework in a particular scenario. We explain these steps in more detail, provide an example which helps better understand how the system works, and report on a series of experimental results. We do not describe steps 3, 4, and 7 of the methodology since they are more technical.

Application domain

With respect to the meta-model of the Implicit Culture terms, in our system, *agents* are software developers willing to find a web service. The working scenario is as follows: an agent submits a *request* for *web service operation* to the IC-Service, which returns a list of recommended services. The request is represented as an *object* and it contains a textual description of the *goal*, the *name of the* desired *operation*, the description of its *input/output parameters*, the *description of* a desired *web service* and an optional list of preferred features (provider, etc.), all represented as attributes. The request is an object of the *submit_request* action. The feedback is collected via the optional *provide feedback* action, which expresses the level of the agent's satisfaction with the result, or via the *invoke* action, which marks a service as suitable for the request. If the agent decides to use one of the services, further information is acquired. The *get_response* action marks a service as available and the *raise_exception* action signals that the service is not available or faulty. Having received the response message, the application can generate a feedback based on extra-knowledge about the expected result: e.g., the feedback is positive if the correct output has been obtained.

An example of a *scene* could be a set of actions corresponding to the invocations of various service operations:

{…; *invoke*(getWeatherByZip(…, *service*=DOTSFastWeather);…), *invoke*(getWeather(…, *service*=GlobalWeather);…)}.

An example of a performed action could be

invoke(Peter; getWeatherByZip(*service*=DOTSFastWeather);;25-Jun-07-14:22),

which states that Peter invoked the operation getWeatherByZip of the DOTSFastWeather web service 25/06/07 at 14:22. In this example, the culture can contain the information which services usually are invoked by a group of service clients for getting a weather forecast.

Cultural theory

The IC-Service processes the request from the system in two steps. In the first step, the *submit_request* action is matched with the theory to determine the next action that must follow, i.e. the *invoke* action. In the second step, the SICS finds situations where the *invoke* action has been previously performed, determining web service operations used for similar requests in the past. In this step, the similarity between the current agent's request and the previously submitted requests is calculated. As a result, the IC-Service returns a set of services that have been used for similar requests in the past. The cultural theory rule can be written as follows:

if *submit_request*(request-X) then *invoke*(operation-Y(service-Z), request-X).

This means that the *invoke* action must follow the *submit_request* action and both actions are related to the same request. The IC-Service cultural theory definition language can be used to specify other requirements as well. For example, the following rule

if *submit_request*(request-X(cost= "*low*")) then *invoke*(operation-Y(service-Z), request-X) ∩

provide_feedback(operation-Y(service-Z), cost = "low")

means that if a service with a low cost is requested, the system will recommend services considered cheap by other clients.

Creating recommendations

The recommendation process consists of the following steps:
Step 1. Find the rule of the theory that matches with the current observation.
Step 2. Find the corresponding cultural action.
Step 3. Find the set of scenes where the cultural action is likely to be performed.

In the following we explain these steps. When an agent performs the submit_request action, the observation corresponding to the action is passed into the SICS where it is matched with the antecedent part of the theory. The information about the request in the observation is used to instantiate variables in the consequent of the rule. The corresponding cultural action could be, for instance, invoke(..., request-X). At the next step, for the given cultural action the set of agents that performed similar actions and the set of scenes where actions have been performed are found. Then a set of agents most similar to the agent submitted the request is selected and the set of scenes is updated accordingly. The similarity of agents is calculated on the basis of their past actions. Finally, scenes where the cultural action is most likely to occur are selected and web services from the scenes are recommended to the originator of the request.

Similarity configuration

Similarity between observed actions (such as submit_request, invoke, etc.) is determined by the similarity of their names, attributes and objects. See (Birukou et al., 2007a) for more details of the similarity calculation algorithms.

Example

Let us illustrate how the search process takes place in practice. Suppose the following request is submitted:

goal : get weather forecast for Rome, Italy;

operation : get weather;

input : city name, country name;

output : weather forecast (temperature, humidity, etc.).

The SICS matches the request action with the antecedent of the theory, and searches for scenes where the invoke action has been performed. Suppose that it finds the following situations:

invoke(...; getWeather (*service* = GlobalWeather), *goal* = get weather report for all major cities around the world; ...);

invoke(...; conversionRate (*service* = CurrencyConvertor), *goal* = get conversion rate from one currency to another currency; ...);

invoke(...; getWeatherByZip (*service* = DOTSFastWeather), *goal* = return the weather for a given US postal code; ...).

The SICS recommends to invoke services which have been considered relevant to the requests previously observed by the system and most similar to the current request. Thus, the getWeather operation of the GlobalWeather web service and the getWeatherByZip operation of the DOTSFastWeather web service can be recommended in response to the request in our example. Suppose that having analyzed the proposed results, the agent invokes the former operation. After observing the invoke action, the SICS will mark this service as suitable for the submitted request, and, in particular, to lexical terms occurred in it. Now, given a request for a service providing information about Italy, the system is likely to suggest the GlobalWeather web service.

Experimental Evaluation

The goal of our preliminary experiments is to evaluate the performance of the system in terms of precision, recall and F-measure:

$$Precision = \frac{Relevant \cap Retrieved}{Retrieved} \quad Recall = \frac{Relevant \cap Retrieved}{Relevant} \quad F = \frac{2 * Precision * Recall}{Precision + Recall}.$$

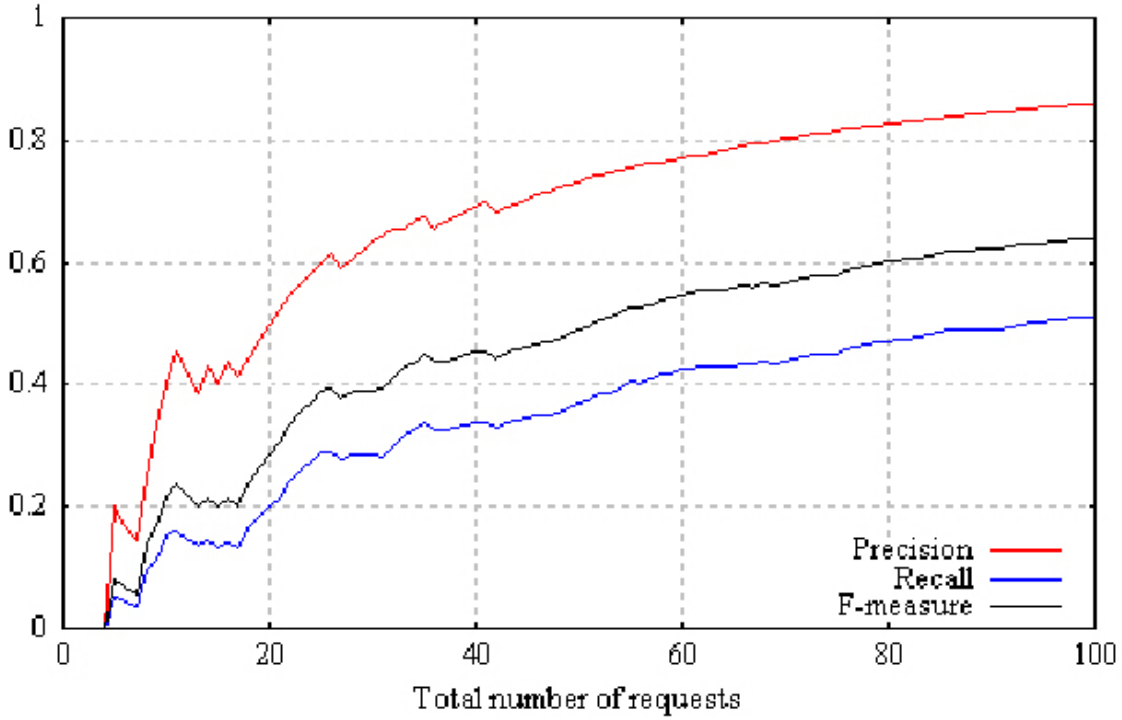
The precision measures the fraction of relevant items among those recommended. The recall measures the fraction of the relevant items included in the recommendations. The F-measure is a tradeoff between these two metrics.

In the experiment we used a collection of 20 web services from xMethods service registry4 divided into 5 topic categories. For each category we found 4 semantically equivalent operations and formed 20 requests based on their short natural language descriptions from WSDL files.

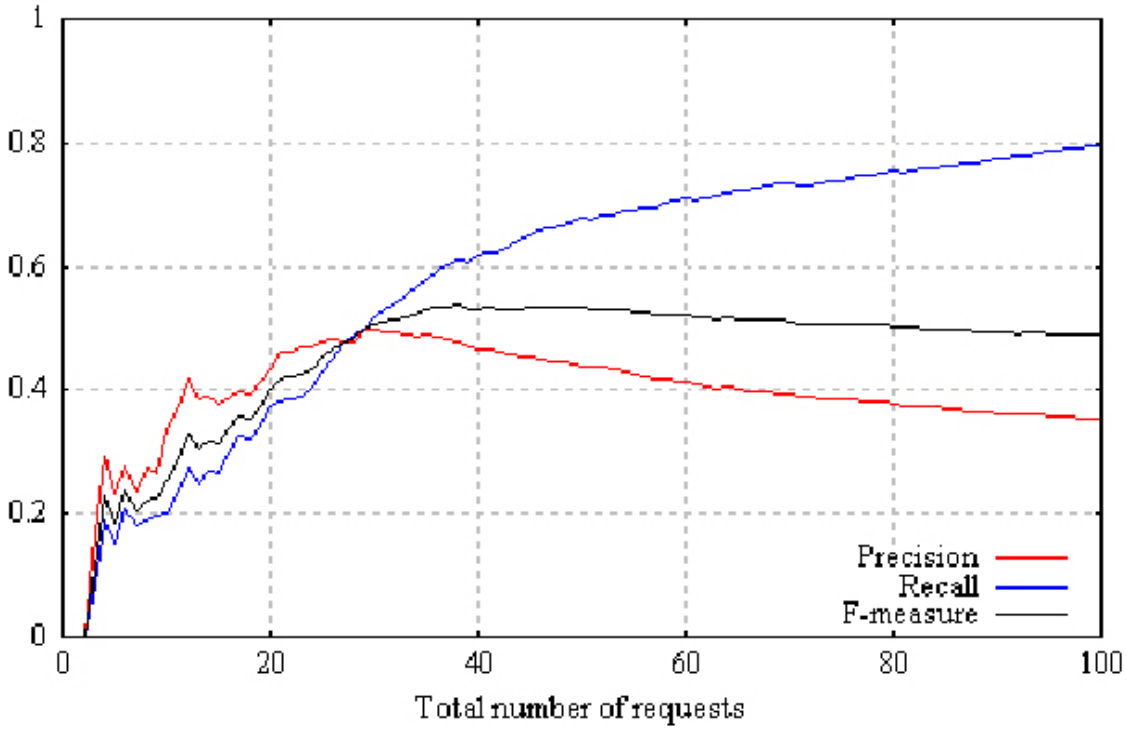
We have defined user profiles to simulate the behavior of real users. A user profile contains a set of requests and a set of web service operations relevant to these requests. The request-generation behavior of the user is simulated by choosing and submitting one of the requests randomly. The result-selection behavior of the user is simulated by choosing and invoking one of the service operations to perform the task. The intuition behind the user profile is as follows: the user submits a request for a service operation. After getting suggestions, (s)he will invoke one of the operations (s)he considers

[4] http://xMethods.com/

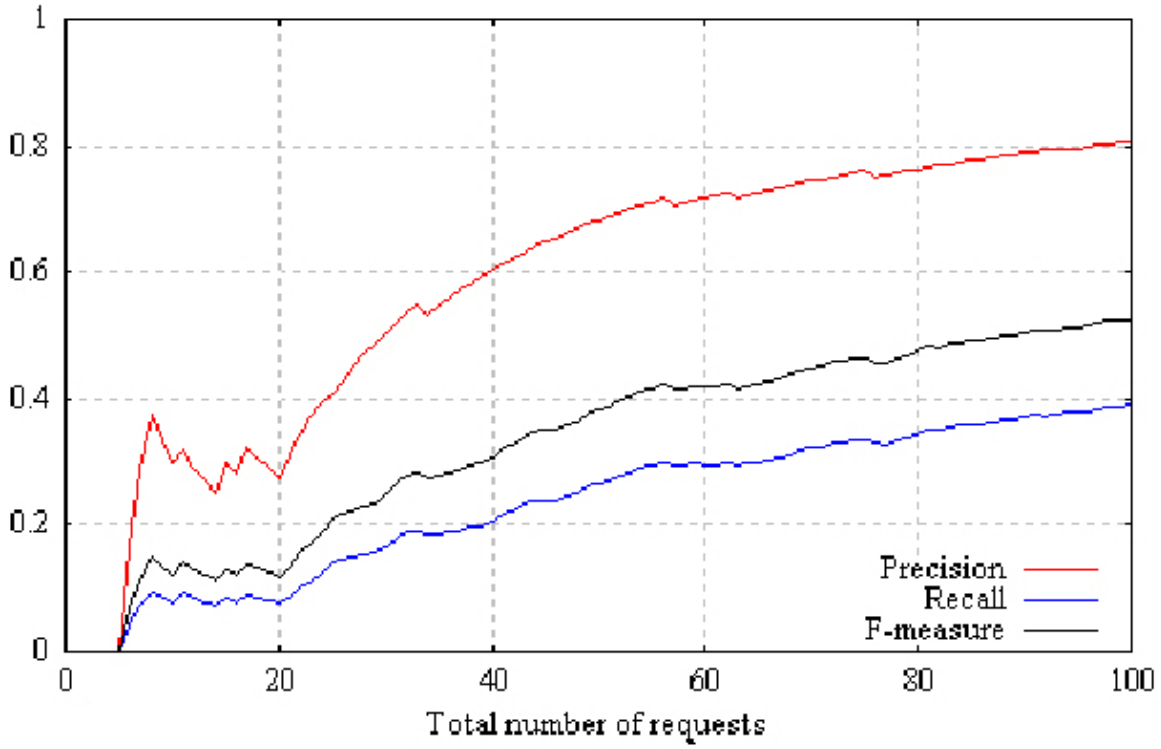
relevant. This invocation is monitored by the SICS Remote Client. During the simulation a random selection is used for choosing which user submits a request to the system in a given moment.



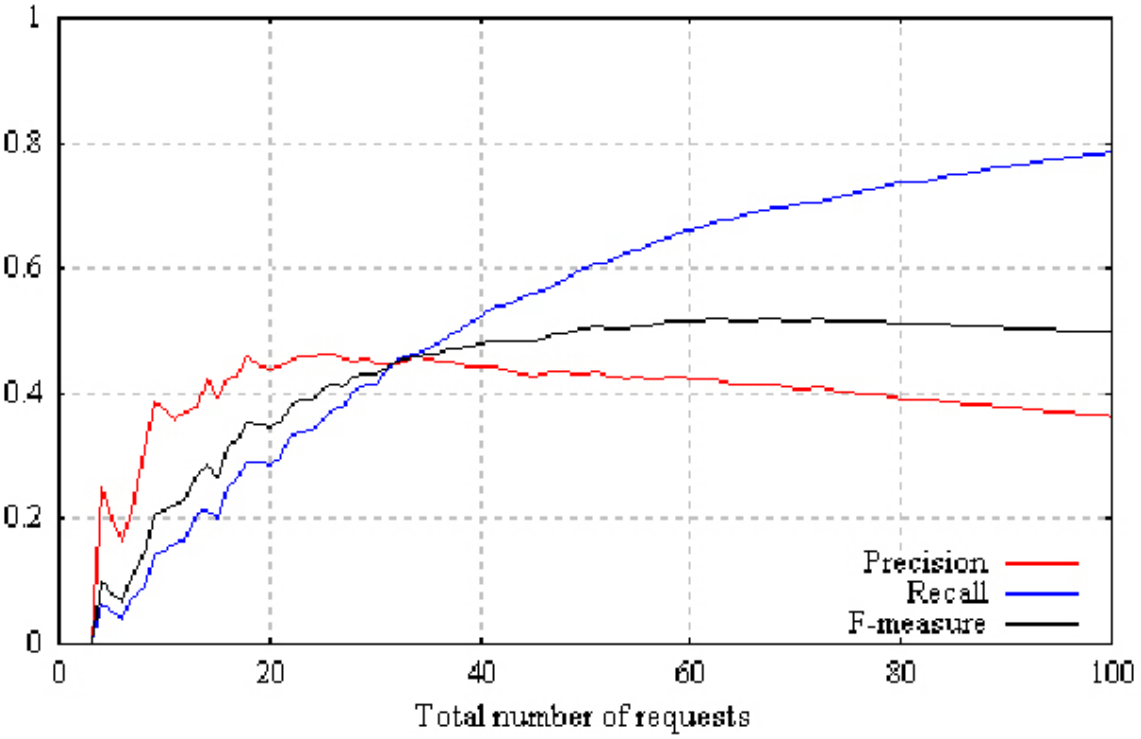
(a)



(b)



(c)



(d)

Figure 13. System performance in four scenarios: (a) VSM with TF-IDF, four clients; (b) WordNet-based semantic similarity metric, four clients; (c) VSM with TF-IDF, 20 clients; (d) WordNet-based semantic similarity metric, 20 clients.

The results for 100 requests submitted to the system are given in Figure 13. The precision, recall, and F-measure of the recommendations are shown. According to these results, the performance tends

to increase with the number of user requests in the case of the TF-IDF similarity metric. Precision of the system with the WordNet-based semantic similarity metric slightly decreases after some point because of faulty positive recommendations produced by the system due to the too generic nature of the lexical similarity used to match requests. However, the recall of the semantic metric is significantly better than the recall of the syntactic one.

FUTURE TRENDS

Recommendations in the approach presented in this chapter are provided based on some pre-defined recommendation strategy. However, the Implicit Culture Framework supports also learning of recommendation strategies by the Inductive Module of a SICS and then using the learned strategies in the Composer Module of the SICS. This would help to adapt recommendations to the target community without any prior assumptions about the community behavior.

The Implicit Culture Framework is not restricted to application in recommendation systems. The elements of the approach are useful for modeling a community of users of social software or an IT system, and for introducing culture in such systems. Explicit modeling of culture of users of social software can help in studying emerging trends, preserving and transferring culture to newcomers. The discovered community culture can be used for determining the best default configuration of the software used by a community. It can also be used for finding similar communities and recommending to newcomers a community that best matches their interests.

In the presented application we have not measured the degree and the speed of culture transfer, but we are planning to do this as a part of future work.

CONCLUSION

In this chapter we addressed the problem of transferring culture between or within communities. The solution we have proposed and evaluated consists of the formalism for defining and representing culture and the framework for transferring some elements of culture. These two parts of the solution constitute a complex systematic approach that includes engineering aspects and aims at representing, making explicit, and transferring elements of culture.

More specifically, first, we have developed formalism for defining and representing culture of communities. Second, focusing on the problem of behavior transfer (a subset of the more general problem of culture transfer), we have proposed the Implicit Culture Framework, an agent-based framework that includes a meta-model for defining the application domain, an architecture of SICS for behavior transfer, algorithms for achieving the implicit culture relation by using SICS. It also includes the IC-Service, a general-purpose, domain-independent service that implements the SICS architecture and the algorithms, and a methodology for applying the Implicit Culture Framework in practice.

We have applied the proposed approach in the domain of recommendation systems and illustrated how to use it in a system for web service discovery. We have also evaluated the performance of our approach in the web service discovery scenario.

REFERENCES

Agrawal, R. & Srikant, R. (1994). *Fast Algorithms for Mining Association Rules in Large Databases.* VLDB '94: Proceedings of the 20th International Conference on Very Large Data Bases, Morgan Kaufmann Publishers Inc., 487-499.

Axelrod, R. (1997): *The dissemination of culture: A model with local convergence and global polarization*. The Journal of Conflict Resolution 41 (2), 203–226.

Baumard, P. (1999). *Tacit knowledge in organizations.* London & Thousand Oaks: Sage .

Bender, S. & Fish, A. (2000). *The transfer of knowledge and the retention of expertise: the continuing need for global assignments.* Journal of Knowledge Management, Emerald Group Publishing Limited, 125-137.

Birukou, A. (2009). *Implicit Culture Framework for behavior transfer. Definition, implementation and applications.* Unpublished doctoral dissertation. University of Trento.

Birukou, A., Blanzieri, E., D'Andrea, V., Giorgini, P., Kokash, N. & Modena, A. (2007). *IC-Service: A Service-Oriented Approach to the Development of Recommendation Systems.* Proceedings of ACM Symposium on Applied Computing. Special Track on Web Technologies, 1683-1688.

Birukou, A., Blanzieri, E., D'Andrea, V., Giorgini, P. & Kokash, N. (2007a). *Improving Web Service Discovery with Usage Data.* IEEE Software, IEEE Computer Society Press, *24*, 47-54.

Birukou, A., Blanzieri, E. & Giorgini, P. (2005). *Implicit: An Agent-Based Recommendation System for Web Search.* AAMAS '05: Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems, ACM Press, 618-624.

Birukou, A., Blanzieri, E., Giorgini, P. & Weiss, M. (2006). *A Multi-Agent System For Choosing Software Patterns.* Technical report DIT-06-065. University of Trento.

Birukou, A., Blanzieri, E., Giorgini, P. & Giunchiglia, F. (2009). *A formal definition of culture.* MICON '09: Proceedings of the Workshop on Modeling Intercultural Collaboration and Negotiation at the Twenty-first International Joint Conference on Artificial Intelligence, 10-24.

Birukou, A., Blanzieri, E., Giorgini, P. & Giunchiglia, F. (2009a). *A formal definition of culture of a set of agents.* Working paper. Retrieved September, 1, 2009 from http://disi.unitn.it/~birukou/publications/Culture.pdf.

Blanzieri, E., Giorgini, P., Massa, P. & Recla, S. Batini, C., Giunchiglia, F., Giorgini, P. & Mecella, M. *(ed.)* (2001). *Implicit Culture for Multi-agent Interaction Support.* CooplS '01: Proceedings of the 9th International Conference on Cooperative Information Systems, Springer-Verlag, *2172*, 27-39.

Bova, R., Paik, H., Benatallah, B., Zeng, L. & Benbernou, S. (2007). *Task Memories and Task Forums: A Foundation for Sharing Service-based Personal Processes*. Proceedings of the International Conference on Service Oriented Computing (ICSOC).

Brusilovsky, P., Farzan, R. & Ahn, J. (2005). *Comprehensive personalized information access in an educational digital library*. JCDL '05: Proceedings of the 5th ACM/IEEE-CS joint conference on Digital libraries, ACM Press, 9-18.

Burke, R. (2002). Hybrid *Recommender Systems: Survey and Experiments*. User Modeling and User-Adapted Interaction, Kluwer Academic Publishers, 12, 331-370

Carley, K. (1991): *A theory of group stability*. American Sociological Review 56 (3).

Claypool, M., Le, P., Wased, M. & Brown, D. (2001). *Implicit interest indicators*. International Conference on Intelligent User Interfaces, ACM Press, 33-40

Curbera, F., Ferguson, D. F., Nally, M. & Stockton, M. L. (2005). *Toward a Programming Model for Service-Oriented Computing.* ICSOC: Proc. of the 3d Int. Conference on Service-Oriented Computing, Springer, *2172*, 33-47.

Farzan, R. & Brusilovsky, P. (2007). *Community-based Conference Navigator.* Proceedings of SociUM Workshop (1st Workshop on "Adaptation and Personalization in Social Systems: Groups, Teams, Communities") at UM2007.

Ferber, J. (1999). *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Addison-Wesley Longman Publishing Co., Inc.,

Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1995). *Design patterns: elements of reusable object-oriented software.* Addison-Wesley Longman Publishing Co., Inc.

Gongla, P. & Rizzuto, C. R. (2001). *Evolving communities of practice: IBM global services experience.* IBM Syst. J., IBM Corp., *40*, 842-862

Hofstede, D.G. (2001). *Culture's Consequences: Comparing Values, Behaviours, Institutions and Organizations Across Nations*. 2nd ed. Sage Publications, Inc.

Keller, U., Lara, R. & Polleres, A. (2004). *WSMO Web service Discovery*. WSML Working Group. Technical report.

Kerrigan, M. (2006). *Web Service Selection Mechanisms in the Web Service Execution Environment (WSMX)*. Proceedings of the ACM Symposium on Applied Computing (SAC), ACM Press, 1664-1668.

Kuroda, Y. & Suzuki, T. (1991). *Arab students and English: the role of implicit culture*. Behaviormetrica, 29, 23-44.

Lave, J. & Wenger, E. (1991). *Situated Learning: Legitimate Peripheral Participation.* Cambridge University Press.

Linden, G., Smith, B. & York, J. (2003). *Amazon.com recommendations: Item-to-item collaborative filtering*. IEEE Internet Computing, 7(1), 76–80.

Manikrao, U. S. & T.V.Prabhakar (2005). *Dynamic Selection of Web Services with Recommendation System*. NWESP '05: Proceedings of the International Conference on Next Generation Web Services Practices, IEEE Computer Society, 117-121.

Martin, D., Burstein, M., et al. (2004). *OWL-S: Semantic Markup for Web Services.* Web Ontology Working group. Technical report.

Mimnagh, C. & Murphy, M. (2004). *Junior doctors working patterns: application of knowledge management theory to junior doctors training.* Proceedings of the conference on current perspectives in healthcare computing, 42-47.

Mulder, M. B., Nunn, C. L. & Towner, M. C. (2006). *Cultural macroevolution and the transmission of traits.* Evolutionary Anthropology: Issues, News, and Reviews, *15*, 52-64.

Nonaka, I. & Takeuchi, H. (1995). *The Knowledge Creating Company* Oxford University Press, New York.

Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P. & Riedl, J. (1994). *Grouplens: an open architecture for collaborative filtering of netnews.* In CSCW'94: Proceedings of the 1994 ACM conference on Computer supported cooperative work, 175–186.

Seco, N., Veale, T. & Hayes, J. (2004). *An Intrinsic Information Content Metric for Semantic Similarity in WordNet*. Proceedings of the European Conference on Artificial Intelligence (ECAI), IOS Press, 1089-1090

Sengers, P. (1998). *Anti-Boxology: Agent Design in Cultural Context.* Unpublished doctoral dissertation, Carnegie Mellon University.

Shaw, M. L. G. & Gaines, B. R. (1999). *Supporting Modeling of the Social Practices of other Users in Internet Communities.* UM '99: Proceedings of the 7th International Conference on User Modeling.

Sherchan, W., Loke, S. W. & Krishnaswamy, S. (2006). *A Fuzzy Model for Reasoning about Reputation in Web Services*. Proceedings of ACM Symposium on Applied Computing (SAC), ACM Press, 1886 – 1892.

Smyth, B., Balfe, E., Freyne, J., Briggs, P., Coyle, M. & Boydell, O. (2005). *Exploiting Query Repetition and Regularity in an Adaptive Community-Based Web Search. Engine* User Modeling and User-Adapted Interaction, Kluwer Academic Publishers, *14*, 383-423.

Spanoudakis, G. & Constantopoulos, P. (1996). *Elaborating Analogies from Conceptual Models.* Applied Artificial Intelligence, *10*, 281-306.

Tremblay-Beaumont, H. & Aïmeur, E. (2005). *Jukeblog: A recommender system in the music weblogs*. Proceedings of the IADIS Int. Conference on e-Commerce, 274–280.

Wenger, E. (1999). *Communities of Practice: Learning, Meaning, and Identity*. Cambridge University Press.

Wenger, E. *Communities of practice a brief introduction.* Retrieved September 1, 2009 from http://www.ewenger.com/theory/

KEY TERMS & DEFINITIONS

Behavior transfer: acquiring by a community the behavior that already widely spread in another community

Community: a set of people who have similar interests or concerns and regularly interact to learn/share information about the subject of their interests or concerns

[Cultural] trait: a characteristic of human societies that is potentially transmitted by non-genetic means and can be owned by an agent

Culture: a set of traits that are shared by the community and are transmitted

Implicit culture relation: a relation between two sets of agents, G and G', and a culture T, such that the agents from G have culture T and the agents from G' acquire culture T without performing explicit actions aimed at acquiring culture T.

Recommendation system: a system that produces recommendations about interesting items in response to the implicit or explicit information requests from its users.

SHORT BIO

Aliaksandr Birukou is a PostDoc in the University of Trento's Department of Information Engineering and Computer Science. His research interests are in recommendation systems, communities, culture, compliance management and his professional experience includes banking software development. He received his PhD in Information and Communication Technologies from the University of Trento (2009).

Enrico Blanzieri was born in Ferrara, Italy, in 1965. He received the Laurea degree (cum laude) in electronic engineering from the University of Bologna in 1992, and a Ph.D. in cognitive science from the University of Turin in 1998. From 1997 to 2000 he was researcher at ITC-IRST of Trento, and from 2000 to 2002 he was assistant professor within the Faculty of Psychology of the University of Turin. Since 2002 he is assistant professor at the Department of Information Engineering and Computer Science of the University of Trento. His research interests include soft computing, artificial intelligence and bioinformatics. He co-authored more than 60 publications with 14 papers published in peer-reviewed international journals and 25 papers in peer-reviewed international conferences.

Paolo Giorgini is leading the Software Engineering and Formal Methods group at the Department of Information Engineering and Computer Science of University of Trento. He received his Ph.D. degree from Computer Science Institute of University of Ancona - Italy - (1998) and then he joined the University of Trento as pos-doc researcher. He has been also researcher visiting at the Computer Science Department of University of Toronto (2000) and at the Software Engineering Department of University of Technology in Sydney (2005). He has worked on the development of requirements and design languages for agent-based systems, and the application of goal-oriented techniques to software analysis and development. He is one of the founders of Tropos, an agent-oriented software engineering methodology. He is Co-editor in Chief of the International Journal of Agent-Oriented Software Engineering (IJAOSE) and his publication list includes more than 150 refereed journal and conference proceedings papers and eight edited books.