**International Doctorate School in Information and
Communication Technologies**

DISI - University of Trento

Implicit Culture Framework
for behavior transfer.
Definition, implementation and applications.

Aliaksandr Birukou

Advisor:

Prof. Paolo Giorgini

Università degli Studi di Trento

Co-Advisor:

Prof. Enrico Blanzieri

Università degli Studi di Trento

Abstract

People belong to different communities: business communities, Web 2.0 communities, religious communities, scientific communities, just to name a few. Everyone can belong to and acquire experience in more than one community. This experience is related to the community activity and comes in the form of best practices, behavior, implicit (tacit) knowledge, ways of using artifacts, etc. All these accumulates and evolves over time and slowly becomes a part of the culture of the community. If community activity is very specific, it can be reflected also in the specificity of the community culture. Newcomers in such community might suffer from what is called "culture shock", i.e. a feeling of confusion when not able to grasp what is common for old-timers. This occurs because part of the community culture is not explicit, i.e. not readily available, and it is very hard to extract valuable information from it. Such information can be used for increasing economic and social benefits of the community members (e.g., for performing recurring tasks, easier integration of newcomers, better quality of life). Moreover, the awareness of the community culture could help the community to handle the turnover of members and structural changes, while preserving the culture. All these introduce the need for transfer of culture between or within communities.

Currently, there is no domain-independent approach for discovering, representing, transferring, and preserving community culture. Moreover, taking into account the amount of information accumulated by communities, computer aided tools for such representation and transfer are of utmost importance. A key property of such tools should be their non-intrusiveness, i.e. they must be as much integrated in the community practices as possible. Research challenges in solving these problems include, but are not limited to: 1) providing a generic approach for dealing with community culture; 2) designing a framework and computer aided supporting tools for transferring culture; 3) implementing the framework, applying and evaluating it in different domains.

This thesis addresses the problem of culture transfer. First, we formalize the notion of culture, which includes behavior, knowledge, artifacts, best practices, etc., and provide a classification of problems that involve culture. 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. Then we describe three applications developed using the IC-Service in the domain of recommendation systems: a system for web search, a system for software pattern selection, and a system for web service discovery. Finally, we present the results of the evaluation of the applications with real users and with ad-hoc user models.

Keywords
culture, communities, behavior transfer, agents, recommendation systems

Contents

List of Acronyms

CBR   Case-Based Reasoning

FIPA   Foundation for Intelligent Physical Agents

JADE   Java Agent DEvelopment framework

MAS   Multi-Agent System

SICS   System for Implicit Culture Support

Chapter 1

Introduction

1.1 Motivation

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" [152]. People in a community of practice interact and develop shared competence and experience related to their activity [153]. The accumulated experience is probably the most important result of the community interactions and it comes in the form of behavior, best practices [125], ways of using community artifacts [88] and addressing recurring problems [152], and implicit or explicit knowledge [12, 106]. 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. Benefits and impact achieved from more effective use of the community culture are not always the same, but, rather, depend on the community. For instance, sharing experience of university professors in writing grant proposals would attract more money to the university. For the users of a system, learning best practices and usage patterns would help them to use the system in a more efficient way. Dimitrova et al. [42] states unawareness of current trends in the community, and difficulties in finding users' role in the group among possible difficulties in online communities of users of social software. According to the authors, these difficulties ultimately reduce "[...] the effectiveness of the community to create, share, evaluate and evolve knowledge". Moreover, in many cases it is critical to preserve the community culture in spite of the turnover of members and other changes in the community structure. For instance, the software release process should not depend on the people currently working in the company [132]. Finally, some communities can benefit from acquiring the culture of another community. For instance, a university network administrator has to

acquire the knowledge about peculiarities of the network; PhD students would like to use the knowledge of more senior members of the research group about the state-of-the-art; a researcher that needs to go to a conference could ask colleagues for suggestions about hotels and airlines flying there.

1.2   The problem

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, as we have already pointed out, 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 [14], leveraging company's knowledge [106], sharing implicit knowledge in communities of practice [60, 102].

Different approaches address some aspects of the above-mentioned problem. Nonaka and Takeuchi [106] 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 [88] as an approach that facilitates acquiring of existing sociocultural 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 [130], in using educational resources [26, 51], etc. Knowledge management in general deals with organizing, creating, capturing and transferring knowledge, trying to ensure its availability for future users. However, as we discuss in Chapter 2, knowledge management approaches mainly focus on the "codifiable" part of the body of knowledge. Also, the notion of culture is broader than the notion of knowledge, thus the problem of culture management is broader than the problem of knowledge management.

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. On one hand, a conceptual framework should be introduced to represent and transfer elements of culture. On the other hand, software systems that automate such transfer should be developed in order to effectively manage information about culture. An important property of such systems should be their non-intrusiveness, in other words they must be as much integrated in the community practices as possible.

1.3 Challenges and objectives

The problem of discovering, representing, transferring, and preserving community culture raises a number of challenges. Such challenges include but are not limited to:

C1. How to determine the scope of the community culture, i.e. what is the content of culture and how to distinguish the culture of the community from cultures of community members?

C2. What are the causes and means of spreading the community culture?

C3. How to transfer some elements of culture, while preventing the transfer of some other element?

C4. How to manage community culture so as to keep it within certain bounds or preserve certain aspects?

C5. How to provide methodologies, computational models and software tools for discovering, representing, transferring, and preserving community culture?

C6. How to develop software for communities taking into account their cultures?

We refine some of the listed challenges into more specific objectives of the thesis (in parentheses for each objective, we specify the related challenges):

O1. How to define a culture of a community? (C1) Moreover, how to provide an operational definition that can be applied to practical problems, including computation and measurement of culture? (C6)

O2. How to devise an engineering approach for discovering, representing, transferring, and preserving community culture? (C5)

O3. How to design and implement the architecture of a general-purpose framework supporting the approach? (C5)

O4. How to develop computer aided tools, supporting the framework, for discovering, representing, transferring, and preserving elements of culture? (C5)

O5. How to apply the tools in practice, for instance, for developing systems? (C6)

1.4 The solution

With respect to the stated objectives, the thesis is developed as follows.

First, we 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 definition and formalization of the notion of community culture and the classification of problems that involve culture address objective O1.* Consistently with the literature, we define traits as "characteristics of human societies that are potentially transmitted by non-genetic means" [103]. Behavior, beliefs, knowledge, norms, rules, values mentioned by many authors as elements of culture, in our formulation are just particular kinds of traits. Traits also include community artifacts, habits, best practices, etc. The list of traits given here is not exhaustive. If something is seen as a potential culture element, is not innate (the requirement of being transmitted by non-genetic means), can be owned by an individual and shared by the community members, it can be classified as a trait. 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. Apart from the definition of culture, in this part of the thesis, we propose a classification of problems that involve culture and occur in various research and application domains.

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. *The Implicit Culture Framework addresses objectives O2 and O3.* It includes a meta-model for defining the application domain, a general architecture of System for Implicit Culture Support (SICS) for behavior transfer, a detailed architecture of SICS modules, algorithms for their functioning, an implementation and guidelines for applying the architecture in practice. We define implicit culture relation as a relation between a set and a group of agents such that the elements of the set behave according to the culture of the group. The SICS architecture performs the transfer of behavior required for achieving the implicit culture relation.

Third, we describe the IC-Service, a general-purpose and domain-independent implementation of the SICS architecture and algorithms. It was developed using modern technologies, such as web services and JavaBeans. *The IC-Service addresses objective O4.* The IC-Service supports behavior transfer that is specified by a pre-defined configuration for the domains that can be represented using the concepts used in the Implicit Culture Framework, namely the concepts of agent, action, object and attribute.

Finally, using the proposed solution we present three applications based on the Implicit Culture Framework in the domain of recommendation systems: a system for web search, a system for software pattern selection, and a system for web service discovery. *The developed applications address objective O5.*

1.5   Contribution of the thesis

The thesis improves the state of the art in several directions.

First, the notion of culture of a community is formally defined in an operational way, and an engineering approach dealing with culture is proposed. This includes discovering, representing, transferring, and preserving culture of a community. Using the proposed formalism it is possible to compute and measure culture in different scenarios and to develop applications adapted to the culture of their users.

Second, a classification of problems involving culture is proposed. The classification helps to treat such problems in a systematic way, e.g. for finding generic problems occurring in different domains and then providing a common solution.

Third, we propose the Implicit Culture Framework that includes a meta-model, the SICS architecture for behavior transfer, a general-purpose and domain-independent implementation of the SICS architecture, and a methodology for its deployment. The Implicit Culture Framework implements an engineering approach for behavior transfer between or within agent communities. Thus, it addresses the need for approaches that transfer culture.

Fourth, we would like to emphasize the development of the IC-Service. The IC-Service

has been applied in different applications, some of them are presented in the following chapters [18, 19] and some of them are out of the scope of this thesis [15, 111, 133], although we briefly mention them in Chapter 8.

Finally, three applications of the proposed approach in the area of recommendation systems are developed and evaluated. The applications illustrate that our approach can be applied and improves quality of recommendations in different domains.

1.6 Structure of the thesis

The thesis has the following structure: in Chapter 2 we present the state-of-the-art that include research on culture and sociality in computer science, implicit and explicit knowledge in knowledge management, and existing approaches for transferring behavior, implicit knowledge, and culture in various disciplines.

In Chapter 3 we review the notion of culture as it comes in anthropology and social science, and propose a formal definition of culture of a set of agents. We then add the temporal dimension to consider dynamics and evolution of culture and propose a classification of problems that involve culture. We conclude that chapter with a set of measures for comparison and assessment of cultures of different communities or cultures of the same community in different moments of time.

In Chapter 4 we consider the problem of culture transfer in terms of the proposed formalism and introduce a narrower problem of transferring such important aspect of culture as behavior. We then describe the Implicit Culture Framework, an agent-based framework for behavior transfer within or between communities and we argue that transferring behavior can lead to knowledge and experience transfer. We present a general architecture of a SICS, which implements the behavior transfer. We then describe a detailed architecture of SICS and algorithms we use inside the architecture. We continue this chapter with the description of the IC-Service, an implementation of the SICS architecture and present a methodology of using the Implicit Culture Framework in different scenarios.

In Chapter 5 we present three applications of the Implicit Culture Framework in the domain of recommendation systems. Section 5.1 describes Implicit, a recommendation system for web links. The IC-Patterns system, a system for recommending software patterns in communities of software developers and architects, is presented in Section 5.2. An application of the Implicit Culture Framework to web service discovery is provided in Section 5.3. For each application we first introduce the reader to the domain, then show how the domain is formulated in terms of agents, objects, actions and attributes, used in the Implicit Culture Framework, and then describe the system. In Section 5.4 we compare the applications.

Chapter 6 contains the description of the objectives, methodology, and results of the evaluation of the developed applications. The measures used in the evaluation include performance, scalability and quality of recommendations.

Chapter 7 overview related work in the following research areas: recommendation systems, collaborative filtering, stigmergy, social navigation. For each area we show the similarities and differences between the area and our approach.

The conclusions of the thesis are given in Chapter 8.

Chapter 2

State of the Art

In this chapter, we review the state of the art. We start with the discussion of computer science approaches that address culture and sociality in Section 2.1. Then we present the notions of implicit and explicit knowledge from the literature on knowledge management, and briefly discuss the implicit-explicit distinction for culture in Section 2.2. In line with our focus on transfer, in Section 2.3, we review existing approaches for transferring behavior, implicit knowledge, and culture. Finally, we conclude this chapter in Section 2.4

2.1 Culture in computer science

Carley [31] 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. This work is further extended in the Construct project [74, 73]. For instance, one of the recent applications of Construct studies the effects of different methods of information diffusion on spreading beliefs and knowledge about illegal tax schemes in different American cities [72]. With respect to the definition of culture we give in Chapter 3, this model of information diffusion is complementary, because it models transmission of elements of culture (e.g., beliefs, knowledge) in a society.

Axelrod [7] 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 and 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.

Epstein and Axtell [47] study the emergence of the group rules from local ones defined at an agent's level in an artificial society of simple agents living and consuming sugar in an artificial environment called "Sugarscape". The authors consider a culture of the society as a string of binary cultural attributes and model cultural transmission both on horizontal (between agents) and vertical (through generations) levels using simple rules. However, they do not provide any formal definition of culture since the main focus of the book is on the emergence of group rules from the local ones.

According to O'Reilly [109], the culture of an organization is considered as strong if wide consensus exists about the content and participants believe in the importance of the content. They also formulate this as a [not necessarily big] set of values that are widely shared and strongly held. This is similar to the notion of strong culture, i.e. culture shared by all pairs of agents in a group, we consider in our formalism.

Some approaches in the field of autonomous agents and multi-agent systems address the issues of sociality in agent societies. Castelfranchi [33] elaborates the concept of social action and stresses that it cannot be reduced to communication. In his view, communication is just a particular kind of social action, and is used by agents because agents are social. Ossowski [110] proposes social coordination architectures and social interaction strategies for decentralized coordination in multi-agent systems. Shoham and Tennenholtz introduce the concept of social laws in multi-agent environments [128]. They define social laws as norms that restrict agent activities so as to achieve dynamically acquired goals while not interfering with other agents. They further extend this work in the direction of co-learning [127], when several agents simultaneously try to adapt to the behavior of each other in order to reach a desirable state of the system. More recent work of the authors studies the emergence of social conventions in agent societies in stochastic settings [129]. Masolo et al. [95] provide a framework for providing a foundational ontology of socially constructed entities. They consider not only social individuals, but also social concepts, such as social roles. Omicini et al. [107] propose the use of the notion of artifact to represent reactive entities (as opposed to pro-active entities, i.e. agents) in multi-agent systems. In particular, artifacts can be used to address social issues of coordination, organization, security in multi-agent systems. The application of artifacts to coordination in multi-agent systems is discussed, for instance, in [108].

However, the issue of sociality alone does not help neither to understand what differentiates one set of agents from another nor to grasp what are the peculiarities of the behavior of agents of a specific society. Although in two different agent societies agents can be able to communicate with each other and perform other social actions, these two societies can be very different from each other. We claim that the concept of culture can be used to describe and compare sets of agents.

More in line with our work on formalizing the notion of culture, Balzer and Tuomela [11] study social practices and the dynamics of their maintenance in groups. They define social practices as recurrent collective activities based on collective intentions. The paper focuses on informal, non-normative practices, such as playing soccer on Sundays, going

to sauna on Saturday afternoon, shaking hands, sharing a ride to work. They also note that the maintenance (change, preservation, renewal) depend on the success of a practice. The main contribution of the paper is a mathematical model for the description of social practices and their maintenance in groups.

Our model of culture is not limited to social practices. Moreover, it allows for inclusion of normative practices as well. However, as a consequence, the model of Balzer and Tuomela allows for a richer description of informal social practices. For instance, our model does not permit expressing intentions, but allows operating on manifestations of activities without going into details of underlying intentions. While authors show that success of a social practice is important for its adoption, for our model it is irrelevant whether a trait is successful in some sense. Our model just captures the fact that the trait is a part of culture, no matter how it occurred. The model presented by Balzer and Tuomela is defined for groups and then goes to the individual level, thereby implementing top-down approach. In our model of culture, we start from a set of traits of an individual, consider transmission as an important means of spreading culture, and then go to the culture of a group. Thus, we implement bottom-up approach. Balzer and Tuomela, while requiring sharing of a social practice within a group, and noting the importance of transmission for spreading practice, include transmission into the model only to a certain extent, namely, considering imitation as an example of transmission. Our model of culture allows for different types of transmission as long as there is a predicate that helps to distinguish occurred transmissions.

2.2 Explicit and implicit knowledge

The literature on knowledge management distinguishes two kinds of knowledge: explicit and implicit. Implicit knowledge is often called tacit knowledge, but in this thesis we do not make distinction between the two terms. Taking a "simple dictionary definition" from Hildreth and Kimble [71], we can define *explicit knowledge* as "[...] the knowledge which can be expressed clearly, fully and leaves nothing implied. An example might be knowledge that can be formally expressed and transmitted to others through manuals, specifications, regulations, rules or procedures [...]". *Implicit (tacit) knowledge* is "[...] that which is understood without being openly expressed; it is unvoiced or unspoken. An example might be the knowledge that a native speaker has of a language." Nonaka and Takeuchi, in their influential work on managing knowledge in Japanese companies, define explicit knowledge as the knowledge that "[...] can be articulated in formal language including grammatical statements, mathematical expressions, specifications, manuals, and so forth, [...] something formal and systematic, [...] can be expressed in words and numbers, and easily communicated and shared in the form of hard data, scientific formulae, codified procedures, or universal principles [...] [106, pp. viii, 8]. The implicit knowledge, according to them, is "[...] something not easily visible and expressible. Implicit knowledge is highly personal and hard to formalize making it difficult to communicate or to share with others. Subjective insights, intuitions, and hunches fall into this category of knowledge. Furthermore, implicit knowledge is deeply rooted in an individual's action and experience, as well as in the ideals, values, or emotions he or she embraces." [106, p. 8]. In artificial

intelligence, explicit knowledge of an agent is defined as the knowledge explicitly contained in the formulas in their knowledge base, while implicit knowledge is that which can be derived from those formulas [49].

As we can see, different authors agree on defining explicit knowledge as formally expressed knowledge, while there is no single definition for implicit knowledge. In fact, Gourlay points out that the concept of implicit knowledge is not clearly defined [62, 63] and reviews the use of the concept of implicit knowledge in the literature on knowledge management, artificial intelligence, sociology, and practical intelligence. He lists six main uses of the concept to describe the knowledge of an individual [63]:

1. someone can do something, but apparently cannot give an account;

2. someone claims they feel something of which they cannot give an account, but it is not clear if subsequent events validate the claim;

3. someone can do something, but not give an account at that moment, but can, if pressed, recall the explicit knowledge that was used tacitly when acting;

4. knowledge existing prior to the situation in which it is effective, and due to innate (biological) characteristics;

5. knowledge existing prior to the situation in which it is effective, and due to cultural factors;

6. situations where A knows something that B does not, but where it could be argued A and B share the same practice.

It is also worth noting that some authors make distinction between tacit and implicit knowledge. For instance, Baumard [12], cited in Gourlay [62], defines implicit knowledge as something we might know, but *do not wish to express*. For him, tacit knowledge is something that we know but *cannot express*; it is personal, difficult to convey, and which does not easily express itself in the formality of language and is thus non-communicable. In general, it is hard to map these definitions to the above-mentioned classification of implicit knowledge, but, for instance, use 3 corresponds to Baumard's definition of implicit knowledge, while use 1 corresponds to Baumard's definition of tacit knowledge.

To complete the review of the implicit and explicit knowledge concepts, let us look at the work on the duality of knowledge by Hildreth and Kimble [71]. They define *hard knowledge* as 'codifiable' and *soft knowledge* as less quantifiable, which cannot be captured and stored so easily. Apart from tacit knowledge in Nonaka's sense, soft knowledge includes internalized experience, skills, internalized domain knowledge and cultural knowledge embedded in practice. It is easy to notice that all these aspects of soft knowledge are covered by the classification of implicit knowledge above, thus the definition of implicit knowledge by Gourlay should include the definition of soft knowledge by Hildreth and Kimble. Further describing the concepts of hard and soft knowledge, the authors argue that in most cases one can not make clear distinction between hard and soft knowledge. They suggest that each knowledge item contains some degree of both hard and soft aspects, thus forming a duality of knowledge. In their view, these suggest that both aspects should be taken into account when trying to manage knowledge, while existing

approaches in knowledge management usually able to deal only with hard, i.e. explicit, aspect of knowledge. In our approach, we are trying to focus on both aspects of culture, as introduced below.

2.2.1 Implicit and explicit culture

Going beyond knowledge, some authors suggest that there are explicit and implicit dimensions also in culture. For instance, Kroeber and Kluckhohn [85, p. 157], cited in Kuroda and Suzuki [87], wrote:

> All cultures are largely made up of overt, patterned ways of behaving, feeling, and reacting. But cultures likewise include a characteristic set of unstated premises and categories ("implicit culture") which vary greatly between societies. Thus one group unconsciously and habitually assumes that every chain of actions has a goal, and that when this goal is reached tension will be reduced or disappear. To another group, thinking based upon this assumption is by no means automatic. They see life not primarily as a series of purposive sequences but more as made up of disparate experiences which may be satisfying in and of themselves, rather than as means to ends.

Gillin and Gillin [58], cited in Kuroda ans Suzuki [87], use the terms "overt" and "covert" to refer to the explicit and implicit dimensions of culture.

An interesting finding by Kuroda and Suzuki [87] is that while learning English, i.e. an explicit part of American culture, Arab students also learned some patterned ways of behavior and reasoning, belonging to the implicit part of American culture. In other words, learning language also influences one's identity.

2.3 Knowledge, behavior and culture transfer

In this section we review existing approaches for transferring knowledge, behavior, and culture.

2.3.1 Behavior transfer in AI

The field of transfer learning problems is an AI field related to behavior and knowledge transfer. The goal of transfer learning is to develop methods for using knowledge acquired in a set of source tasks to improve performance in a related, but previously unseen target task [124]. Below we review some existing approaches to transfer learning and compare them with our approach.

Taylor and Stone [140] propose behavior transfer as a novel method that allows a learner trained on one task to learn faster when training on another task with related, but different state and action spaces. The method is a temporal difference learning method, which is a type of reinforcement learning, and it is specific to reinforcement learning problems. The authors show that for some tasks from RoboSoccer domain the behavior transfer method reduces the training time for learners to reach certain level of performance. The training time for behavior transfer is shorter than the training time to learn the task from scratch.

Talvitie and Singh [139] consider a similar problem of reusing knowledge about some tasks in other tasks. In the terms of Markov Decision Process (MDP), the algorithm they propose allows to produce a mapping from the state space of a new problem to the state space of an already known problem. The algorithm has been tested and proved effective in two transfer learning problems: reusing knowledge for a more complex task than the one already learned, and reusing knowledge for the same task, but without considering some inputs (losing the use of some of agent's sensors).

A multi-layered architecture named CAse-Based Reinforcement Learner (CARL) that uses a novel combination of Case-Based Reasoning and Reinforcement Learning is proposed by Sharma et al. [124]. The architecture is applied in the domain of Real Time Strategy games and allows for transferring experience about game tasks. The authors have shown that using the CARL architecture, the agent's capability to perform transfer learning is not just limited to speeding up learning, but can also lead to either better or the same overall final performance in complex scenarios.

With respect to the topic of this thesis, the approaches discussed above focus on narrow problems occurring in the context of multi-agent learning. More specifically, most of them deal with learning some tasks using the reinforcement learning framework. Therefore, the applicability of the approaches is constrained by the applicability of the reinforcement learning. We see the approach proposed in this thesis as a more general, and more applicable to agent societies that involve humans as opposed to artificial agents. However, it is most probable that for the domains that involve only artificial agents, the approaches described above work better as they are more specialized. Another difference is in the way how past experience is used. The key challenge for the behavior transfer approach is mapping a value function from one problem representation to another, typically larger. The key issue in our approach is to determine the similarity between the states of the environment as faced by an agent at the moment and by some other, similar, agent in the past.

2.3.2   Transferring implicit knowledge

Existing approaches for transferring implicit knowledge include implicit learning [63], social learning [153], and socializing [71]. Here, implicit learning is "[...] a cognitive phenomenon in which people acquire new knowledge without conscious intent or awareness [...]" [137]. Social learning can be defined from the books by Wenger [88, 153] as learning by social participation in the community, i.e. by acquiring sociocultural practices of the community and by construction of an individual's identity through the community. Socialization is not defined, but rather used in a general sense, referring to the process of integrating into community so as to behave in a way that others in the group think is suitable.

Nonaka and Takeuchi [106] propose a model for explaining the process of knowledge creation. The model consists of the four steps: socialization, externalization, combination, and internalization. In the first step, socialization, implicit knowledge is transferred between individuals through observation, imitation and practice. Then, in the externalization step, implicit knowledge is translated into documents and procedures by using analogy or metaphor. In the third step, the explicit knowledge is reconfigured by sorting,

adding, combining and categorizing processes and is spread within an organization. In the last step, internalization translates explicit knowledge obtained by individuals into their implicit knowledge. This process repeats over time, which leads to the phenomenon, called "knowledge spiral", that helps knowledge creation and sharing become a part of the culture of an organization.

Hildreth and Kimble [71] point out that the management of hard knowledge is well established in knowledge management field, and many tools and techniques available to support this form of knowledge management. On the contrary, there are no tools for managing implicit knowledge. However, the authors argue that communities of practice should provide the means for transferring implicit knowledge. They also highlight the importance of social aspect of the implicit knowledge.

Van den Hooff et al. [145] show that different authors agree that communities are an effective environment for sharing implicit knowledge. Then, Van den Hooff et al. investigate the impact of ICT on the knowledge sharing in communities of practice. Based on the literature, they develop a theoretical model that identifies possible impacts of ICT on knowledge sharing within a community. They test the model on two communities that use ICT. The results of the investigation show that willingness and ability to share were not found to predict knowledge sharing behavior, while knowledge sharing is directly influenced by identification, trust, communality (shared information bases) and connectivity (ability to communicate independent of time and place). They also show that face-to-face communication is not pre-requisite for trust, even though it helps to develop trust. This suggest that ICT can be used as a tool for knowledge sharing in distributed environments, online communities being an example of those. They conclude that ICT has a positive contribution to knowledge sharing in communities, but this contribution involves a set of complex influences and relationships.

2.3.3 Knowledge and culture transfer in organizations

In the literature on organizations we can distinguish two directions related to our work. The first direction deals with knowledge transfer, while the second direction deals with acquiring organizational culture by newcomers. In this section, we briefly discuss existing literature with respect to these two research directions.

Procedures for successful knowledge transfer help organizations to derive more value from the intellectual assets accumulated within organizations. The benefits of knowledge transfer include increases in performance, adaptation, collaboration and innovation [118].

Schreiber and Carley [118] study the effect of databases on knowledge transfer within organizations. More specifically, they simulate interactions of expert and non-expert agents in an organization with two kinds of databases: task database, which contains knowledge in relation to tasks; and referential database, which contains the knowledge about who in the organization is an expert on a certain topic. The results show that an increase in task complexity leads to a decrease in group performance, while experience improves performance. In addition, they show that the use of task database on simple tasks improve organizational performance, while the use of referential database on these tasks decreases performance. On the other side, the use of referential databases helps to mitigate the drop of performance for non-experts working on complex tasks.

Cataldo et al. [34] studies how breadth of skills, task experience, group experience, and certain environmental attributes affect knowledge transfer within an organization and among organizations. They implement a simulation model based on constructural theory [31]. The results show that skills, task experience, and group experience are important factors affecting how knowledge is transferred in an organization. With respect to inter-organizational knowledge transfer, the results indicate that uncertainty, environmental competitiveness, and breadth of skill are important factors affecting knowledge transfer.

Bender and Fish [14] discuss the transfer of knowledge and expertise in organizations operating on a global scale. They advocate information technology as a necessary tool for knowledge transfer and provide examples of such tools: e-mail, groupware, Internet, intranet, and videoconferencing. They then identify some barriers for knowledge sharing: first, people do not like to share their best ideas, second, people do not like to use other people's ideas, and third, people like to consider themselves experts and prefer not to collaborate with others.

When a new person joins an organization, by reading regulations, norms, etc., they very quickly grasp the explicit part of the culture of the organization. However, the implicit part of organizational culture remains unknown to the newcomer. Examples of elements of such implicit part of organization culture may include the following rules and knowledge items:

- you do not go to the canteen between 12.30 and 13.00 because there is a huge queue;

- even though you can take 28 days of holiday a year, in practice employees take no more than 15 days a year;

Wyatt-Haines [154] formulates the problem of a newcomer in organizational settings:

Culture has been described in many ways by many people:

- A set of guiding beliefs and philosophies
- The way we do things round here
- A way of thinking and acting
- Just the way things are
- The glue that holds the organization together
- Shared beliefs, customs and practices which are often accepted without question

In light of this, it is a strange, yet sad, fact of organizational life that when you join a new organization you are inducted into its systems and processes and introduced to the key people, but it is left to you to learn about the culture. Whereas, if an introduction to the culture was an overt part of the induction process, your ability to fit in and perform in the expected manner would be much accelerated.

Although there exist computational models of organizations (e.g., garbage can [38] and NK [151] models, or, more recent approaches, [32] and [69]), we are not aware of computer science approaches dealing with the problem of culture transfer in organizational settings.

2.4 Concluding remarks

We have reviewed the state of the art with respect to addressing notions of culture, sociality, knowledge in the literature. Existing computer science approaches consider culture as distribution of information, set of features, and string of cultural attributes. As we show in the next chapter, our definition of culture allows for various types of culture content, including the above-mentioned knowledge, features, ideas, and beliefs. Current approaches focus on the means of transmission of culture. Our formalism, as we show in the next chapter, uses the notion of transmission to define culture. On the one hand, we not address the issue about how transmission takes place. On the other hand, in Chapter 4 we propose an approach for transferring some elements of culture. Transfer can be seen as a kind of transmission, but it is usually directed and has some purpose.

We have introduced the implicit and explicit dimensions of culture and knowledge, providing an extensive discussion on the implicit knowledge and showing why it is so hard to capture. Implicit knowledge is addressed in the literature much broader than implicit culture and approaches for dealing with it could provide some insights on how to deal with implicit culture.

Finally, in line with our our focus on transfer, we have reviewed existing approaches for transferring behavior, implicit knowledge, and culture. We have discussed how the problem of behavior transfer is addressed in multi-agent learning, and why communities of practice could become a tool for sharing implicit knowledge.

In the next chapter we review the notion of culture in anthropology and social science and propose a formal definition of culture, emphasizing the aspects of sharing and transmission, while trying to be generic when considering the content of culture.

Chapter 3

Formal Definition of Culture

In the previous chapter, we discussed how culture is addressed in the computer science literature. We start this chapter in Section 3.1, with a discussion of the concept of culture in a variety of disciplines, including anthropology and social science. We also informally introduce basic notions used through this chapter. In Section 3.2, we provide a formal definition of the culture of a set of agents and of the related concepts, while in Section 3.3 we introduce states to investigate the dynamics of culture. Our goal is not to provide a formalism or a reasoning framework per se, but, rather, to give an operational definition of culture that can be used for computing and measuring culture in different scenarios. Therefore, in Section 3.4, we classify the problems that involve culture and occur in various research and application domains. We then define measures for culture in Section 3.5. Through the chapter we use an example of culture of people from a fictitious country to show how our approach can be used to deal also with culture in the anthropological sense. To show that it is suitable for studying culture in Web 2.0 systems and other software, in Section 3.6, we consider a case study of culture in one of Web 2.0 communities. We conclude the chapter in Section 3.7.

3.1 The concept of culture

Culture is a slippery and ubiquitous concept. Initially, culture was associated with the notion of civilization tout-court. At the end of the 30s Margaret Mead put in contrast "culture" with "a culture". "Culture means the whole complex of traditional behavior which has been developed by the human race and is successively learned by each generation"([98] cited in [25]). However, specificity of the notion of culture with respect to a given human society was needed in order to study other societies. So the same citation goes on as: "A culture is less precise. It can mean the forms of traditional behavior which are characteristic of a given society, or of a group of societies, or of a certain race, or of certain area, or of a certain period of time" (cited in [25]). As a consequence, in the anthropological literature culture has been introduced as the concept denoting the object of study of cultural anthropology. Other definitions were proposed and they largely vary. However, they seem to converge to the notion that culture is learned [7], it is associated with groups of people and its content includes a wide range of phenomena including

norms, values, shared meanings, and patterned ways of behaving [109, 99, 97, 24, 23, 85]. In anthropological literature the usefulness of the notion of culture as a scientific tool has been attacked giving rise to the so-called "writing against culture movement" (see Brumann [25] for a reaction against it). The culture as defined in anthropology usually refers to societies defined in national or ethnic terms, however, the concept of culture has been recently used for describing knowledge and behavior of other groups like in the concepts of corporate culture or organizational culture [109, 69, 117]. Moreover, globalization has brought about the problem of interaction of cultures. On the one hand, such interaction leads to blurring boundaries between cultures, while on the other hand it leads to the increasing need of cultural-aware managers and professionals. Recent anthropology textbook definitions take into account the shift in meaning as, for example, in the definition by Peoples and Bailey:

> *Culture is the socially transmitted knowledge and behavior shared by some group of people* (Peoples and Bailey [8, p. 23] cited in [25]).

3.1.1   Culture in historical perspective

Earlier authors define culture in the following ways (cited in Brumann [25]):

- Culture ... refers ... to learned, accumulated experience. A culture ... refers to those socially transmitted patterns for behavior characteristic of a particular social group (Keesing [78, p. 68]).

- Culture, or civilization, ... is that complex whole which includes knowledge, belief, art, law, morals, custom, and any other capabilities and habits acquired by man as a member of society (Tylor [143, p. 1]).

- The culture of any society consists of the sum total of ideas, conditioned emotional responses, and patterns of habitual behavior which the members of that society have acquired through instruction or imitation and which they share to a greater or less degree (Linton [91]).

- A culture is the total socially acquired life-way or life-style of a group of people. It consists of the patterned, repetitive ways of thinking, feeling, and acting that are characteristic of the members of a particular society or segment of a society (Harris [68]).

As we can see, definitions agree on the fact that culture consists of something that is shared and/or learned by a group of people, but the content of the culture varies in different definitions. Similarly to Axelrod [7], we see the content of the culture as a set of traits[1], which can refer to behavior, knowledge facts, ideas, beliefs, norms, etc. In the anthropological literature traits are defined by Mulder as "characteristics of human societies that are potentially transmitted by non-genetic means" [103].

Sperber [135] aims at reconciling the materialist point of view with the study of culture and his solution is that culture is related to mental representations and, consequently, to

---

[1]Traits are further grouped in *features* in Axelrod's formulation, i.e. each feature can take value from a set of specific traits.

physical brain states. In Sperber's view, cultural representation are mental representations which are widely shared within a human group, where "shared" means that individuals belonging to the group have mental representations similar enough to be considered versions of one another. Transmission, according to Sperber, is

> [...] a process that may be intentional or unintentional, cooperative or non-cooperative, and which brings about a similarity of content between a mental representation in one individual and its causal descendant in another individual.

Imitation and communication are listed as two main means of cultural transmission, and Sperber does not agree with defining communication as a coding process that is followed by a symmetrical decoding process and which implies the replication of thoughts in the minds of the audience. He suggests that communication is essentially a transformation process and there are different degrees of transformation ranging from total loss of information to duplication. The same property is advocated for the imitation. From Sperber's point of view, "[...] only those representations which are repeatedly communicated and minimally transformed in the process will end up belonging to the culture" [135, p. 83] and he points out that it is the epidemiology of these representation that should be taken into account. As for transmission, Sperber suggests that the transformations in the process of transmission of representations occur because of the cognitive modularity [136]. It is worth noting that Sperber proposes to study and model the cultural phenomenon as the epidemiology of specific representations and believes that culture is not possible without cognition. So, Sperber emphasized both aspects that we think are important for culture: sharing and transmission.

3.1.2 Defining culture

Looking closer at the definition of culture by Peoples and Bailey [8], it is worth noting that it can be specified as composed of two separate dimensions: 1) knowledge and behavior shared by a group; 2) knowledge and behavior (socially) transmitted. Following Peoples, Bailey, and Sperber, we call "cultural" the traits that respect both the condition of sharing and transmission.

Sharing in our terms means that individuals of a group have the same trait. We would like to stress that some traits cannot be "owned" by individuals and only occur in a group, e.g., low birth rate in a country. Another example is a property of culture of being individualistic, collectivistic, or familistic. Such "society" traits are not covered by our definition of cultural traits. This is not a limitation, in fact, from Sperber's view of representations as residing in minds of individuals it follows that each culture trait can be owned by an individual. Moreover, we believe that such society traits can be added as an extension to our model later, and some of them can be computed from the traits of individuals. To summarize, there are two levels of traits: personal traits and society traits, and at the moment we are modeling only personal traits.

Transmission of a trait in our terms means that the fact that some individual has the trait and some course of actions lead to another individual acquiring the trait. In other words, transmission is when a trait is transferred from one individual to another. Thus, we define culture in the following way:

Culture is a set of traits that are shared and transmitted.

In the following examples, we use knowledge and behavior as particular kinds of traits to show that both sharing and transmission dimensions are important; indeed, we cannot have culture without any of them. The examples are summarized in Table 3.1.

Pasta. As a positive example of culture let us mention pasta in the Italian food culture. The knowledge and behavior related to cooking and consuming pasta are both shared and transmitted.

Mendel's experiments. In other cases, the transmission to some members of the group does not necessarily imply sharing by the group. For instance, even that Mendel's pioneering work in genetics was published in 1866 and it was ignored for the following thirty-five years. In this case, the results were transmitted to some people, and were shared by them, including Mendel himself, but the results were not shared by the general public during Mendel's life, so they can be hardly considered a part of the culture of the scientists at that time.

Explosion. Similarly, sharing of knowledge and behavior without transmission does not constitute culture. For example, let us consider a group of people that witness an extraordinary event, e.g., an explosion. These people share the knowledge about the event, such as when and where it occurred, and may share some behavior such as fleeing, but the knowledge about when and where the explosion occurred and the behavior of fleeing are not culture until they are transmitted. For instance, if all the people who witnessed the explosion die the next day, the knowledge about the explosion will disappear. However, once transmitted, it can be transmitted again and again, and even evolve in a kind of legend, which certainly can be considered a part of culture.

Robinson Crusoe. To give an example of knowledge and behavior that are neither shared nor transmitted, let us consider a group of people from different cultures. As an example of such a group, we could consider fictional characters of Robinson Crusoe and Friday who in the beginning had nothing in common except for being on the same island. However, as the time passed they worked out communication methods and there was even transmission, e.g. of the behavior of salting the food, that lead to shared behavior. This example, apart from showing that no culture is possible without transmission and sharing, shows that the culture can evolve over time and motivates the study of the dynamic aspect of culture we undertake in Section 3.3.

Example	Sharing	Transmission	Conclusion
Pasta	Yes	Yes	culture (all Italians do it and they are told to do it)
Mendel's experiments	No	Yes	no culture before the re-discovery of the experiments
Explosion	Yes	No	no culture
Robinson Crusoe	No	No	no culture

Table 3.1: Culture in the perspective of sharing and transmission dimensions.

3.1.3   Mapping between existing definitions and our definition

Existing definitions of culture are numerous (see, for instance, Brumann [25] for some of them, or Kroeber [85] for even more) and involve different concepts. Here we show

how the concepts that occur in the definitions of culture most often can be related to our definition.

Let us start from the fact that culture is considered to be *learned* (Mead [98],Keesing [78, p. 68] cited in [25]). As we mentioned, a behavior is a particular kind of trait. Thus, in our terms, a learned behavior is either a particular case of a transmitted behavior or a behavior acquired by someone. Learning in the former case could be seen as one of the means of transmissions, together with imitation and communication. In the case of an individual acquiring the behavior alone, the behavior can hardly be considered a part of culture, as we show in the following subsection.

It is often assumed [99], [68, p. 144], [78, p. 68], [91, p. 288], all cited in [25], that culture contains patterns of behavior, i.e. behavior that repeatedly occurs across the society. A pattern of behavior, or a patterned way of behaving is an intensional definition of behavior, so it fits our definition. The culture content is normally considered to be shared by the society members [91, p. 288], [8] and, as highlighted by Brumann, it is very often only partially shared. As we see in the following, our definitions allow for culture that is partially or, in our terms, weakly shared. Norms, rules, values are also named as parts of culture [97, 24], [23, p. 44]. In our terms, norms, rules, values, knowledge and beliefs [97, 143], shared meaning [109] are just particular kinds of traits. The list of traits given here is not exhaustive, and if something is seen as a potential culture element, it can be classified as trait if it is not innate (the requirement of being transmitted by non-genetic means), can be owned by an agent and shared by a set of agents.

3.1.4 Culture and the individual

The relationship between culture and a single individual is twofold. On the one hand, culture forms and changes some personal traits of the individual. On the other hand the individual contributes to the development of the traits that are a part of culture, i.e. cultural traits. We can see the process of the formation and changing of someone's traits as the development of the culture of a specific human being. Without social influence the personal traits of an individual cannot become similar to the culture of the society. For instance, there are serious doubts that feral children [30] are able to develop even their own identity not speaking about culture. Nature against nurture debates [115] also suggest that there is something beyond nature which makes someone human. Looking on the individual-culture relationship from the other side, the culture of a person who stops being part of a society is not evolving in the same way as the culture of the society. Consider Robinson Crusoe whose personal traits were influenced by the cultural traits of the society he belonged to, but since his arrival on the island he had not been contributing to the evolution of the English culture anymore nor he received recent developments of that culture as a result of transmission. All these strongly suggests that one person is not enough to grasp the phenomenon of culture. In the case of languages, this is consistent with Wittgenstein's argument about the fact that it is not possible to have a private language.

Let us consider examples of feral children and Robinson Crusoe in the perspective of sharing and transmission dimensions, using knowledge and behavior as particular kinds of traits. Feral children, for example the fictional character of Tarzan, grown by a group

of monkeys, do not actually share knowledge and behavior with other humans nor they transmitted them. In the case of Tarzan the absence of sharing is even more evident, because there is no group. Since the knowledge and behavior are not shared in the example of feral children, it is impossible to talk about culture in that case. The case of Robinson Crusoe is different, because even being alone, he preserves and tries to transmit, for instance to Friday, the culture he acquired in his home country. While in the case of feral children we can speak about attempts to produce culture, in the case of Crusoe there are attempts to preserve culture.

3.1.5 Culture and two individuals

In case of two individuals, it is possible to speak about culture because it is possible to define sharing and transmission. Similarly to the interpretation given by Sperber, "sharing" in our approach means that two individuals have the same trait. For instance, if both individuals are able to read in English, this trait can be a part of their culture. If one of them is able to speak Japanese and the other one is not, then this behavior is not shared and, consequently, is not a part of their culture. However, if the person who speaks Japanese teaches another one to do it, this is a transmission. Moreover, since they both have the behavior of speaking Japanese and this behavior has been transmitted, this behavior becomes a part of their culture because it is both shared and transmitted. In Robinson Crusoe's case, an example of transmission would be teaching Friday to salt food.

3.1.6 Culture and the group

The examples above show that a culture is tightly coupled with the group. Therefore, we define a culture of a group of people, leaving the definition of culture per se to anthropologists. Let us see how it is possible to extend the ideas of defining culture for two individuals, as presented in previous subsection, for groups that consist of more than two people.

If we consider only two people, it is easy to determine if a trait is shared or not. With more than two people it is not so simple. For instance, let us consider two individuals from the previous example, they both can speak Japanese and English now. Let us imagine that a third person, who speaks and reads only English, joined them. Is the behavior of speaking Japanese still shared, even though there is a person who cannot speak Japanese? Or only the behavior of reading in English, which all three have, is shared? We will address this issue by introducing two definitions. We define *a culture* as traits shared by at least two members of the group and transmitted to at least one member of the group. We define *a culture in a strong sense* as the traits that are shared by everyone and were transmitted to at least one member of the group.

3.1.7 Culture of an individual vs. culture of the group

Although we have shown that it is hard to speak about a culture in case of an individual, we can speak about of the culture of an individual who leaves the group, as Robinson

Crusoe did, or who is a member of several groups. In these settings, individuals receive some information related to the culture of the groups they do not belong currently. It leads to the situation in which someone belongs to the group and has some traits that are part of the culture of the group, and some traits which are not. For instance, while in Italy speaking Italian language is a part of culture, some Italian could speak Japanese and this behavior is not part of culture in Italy. So, the set of all cultural traits that a person has, which could be probably called the culture of the person, is the union between projections of the group cultures the person belongs to. In the example above, it would be the behavior of speaking Italian and the behavior of speaking Japanese, projected, respectively, from the Italian culture and from the culture of the people who attended courses of Japanese or lived in Japan for a while. We should also note that sometimes there are traits that are an essential part of the culture of some society, but cannot be attributed to individuals, as in above mentioned examples of birth rate and individualistic/collectivistic/familistic property of culture. In the following sections we do not consider such society traits and focus only on those cultural traits an individual can possess.

3.2 A formal definition of culture

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" [52]. An agent can have different *cultural traits*, which are characteristics of human societies that are potentially transmitted by nongenetic means and can be owned by an agent. The requirement "can be owned by", which we add to the definition by Mulder [103], 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.

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

Example 1. Let Ag in our example be a set of people: Charlie, Pedro, Maria, and Andrea are European citizens, and Toru is from Japan. Let $\mathcal{T}$ be a set of traits of different types, as shown in Table 3.2. For each trait, we also put its abbreviation (used in the figures in this section) in parentheses.

Table 3.3 lists the set of traits $\mathcal{T}$, and the sets of traits of the specific agents of $Ag = \{Charlie, Pedro, Toru, Maria, Andrea\}$.

We can write $has(Maria, Dante\_Alighieri\_wrote\_Divine\_Comedy)$, or $has(Charlie, cappuccino\_is\_coffee)$, but not $has(Andrea, eating\_with\_sticks)$. We will use this example as a running example through the section ⊘

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 *taking_vacation_in_August* as a norm. We believe that there is no single classification and this suggests that our approach of dealing with generic traits rather than with specific types of cultural content provides certain advantages.

trait type	traits
knowledge	$Dante\_Alighieri\_wrote\_Divine\_Comedy(DA)$, $latte\_macchiato\_is\_coffee(LM)$, $Meiji\_era\_was\_in\_1868\_1912(ME)$, $cappuccino\_is\_coffee(CI)$
behavior	$eating\_with\_sticks(ES)$, $eating\_with\_fork(EF)$, $taking\_vacation\_in\_August(TVA)$, $taking\_vacation\_in\_May(TVM)$
norms, rules	$never\_put\_mayonnaise\_on\_pizza(NP)$, $take\_only\_week\_of\_vacation\_per\_year(T1W)$, $never\_drink\_cappuccino\_after\_lunch(ND)$, $never\_open\_umbrella\_inside\_building(NO)$
beliefs	$Christianity(Chr)$, $Buddhism(Bud)$

Table 3.2: The set of traits $\mathcal{T}$ in Example 1.

set	traits
$\mathcal{T}$	$Dante\_Alighieri\_wrote\_Divine\_Comedy$, $latte\_macchiato\_is\_coffee$, $Meiji\_era\_was\_in\_1868\_1912$, $cappuccino\_is\_coffee$, $eating\_with\_sticks$, $eating\_with\_fork$, $taking\_vacation\_in\_August$, $taking\_vacation\_in\_May$, $never\_put\_mayonnaise\_on\_pizza$, $take\_only\_week\_of\_vacation\_per\_year$, $never\_drink\_cappuccino\_after\_lunch$, $never\_open\_umbrella\_inside\_building$, $Christianity$, $Buddhism$
$T_{Charlie}$	$Dante\_Alighieri\_wrote\_Divine\_Comedy$, $latte\_macchiato\_is\_coffee$, $cappuccino\_is\_coffee$, $eating\_with\_sticks$, $eating\_with\_fork$, $taking\_vacation\_in\_August$, $never\_put\_mayonnaise\_on\_pizza$, $Buddhism$
T_{Pedro}	$Dante\_Alighieri\_wrote\_Divine\_Comedy$, $latte\_macchiato\_is\_coffee$, $cappuccino\_is\_coffee$, $eating\_with\_fork$, $taking\_vacation\_in\_August$, $never\_drink\_cappuccino\_after\_lunch$, $Christianity$
T_{Toru}	$Meiji\_era\_was\_in\_1868\_1912$, $cappuccino\_is\_coffee$, $eating\_with\_sticks$, $taking\_vacation\_in\_May$, $Buddhism$
T_{Maria}	$Dante\_Alighieri\_wrote\_Divine\_Comedy$, $latte\_macchiato\_is\_coffee$, $cappuccino\_is\_coffee$, $eating\_with\_sticks$, $eating\_with\_fork$, $taking\_vacation\_in\_August$, $Christianity$
T_{Andrea}	$Dante\_Alighieri\_wrote\_Divine\_Comedy$, $latte\_macchiato\_is\_coffee$, $cappuccino\_is\_coffee$, $eating\_with\_fork$, $taking\_vacation\_in\_August$, $Christianity$

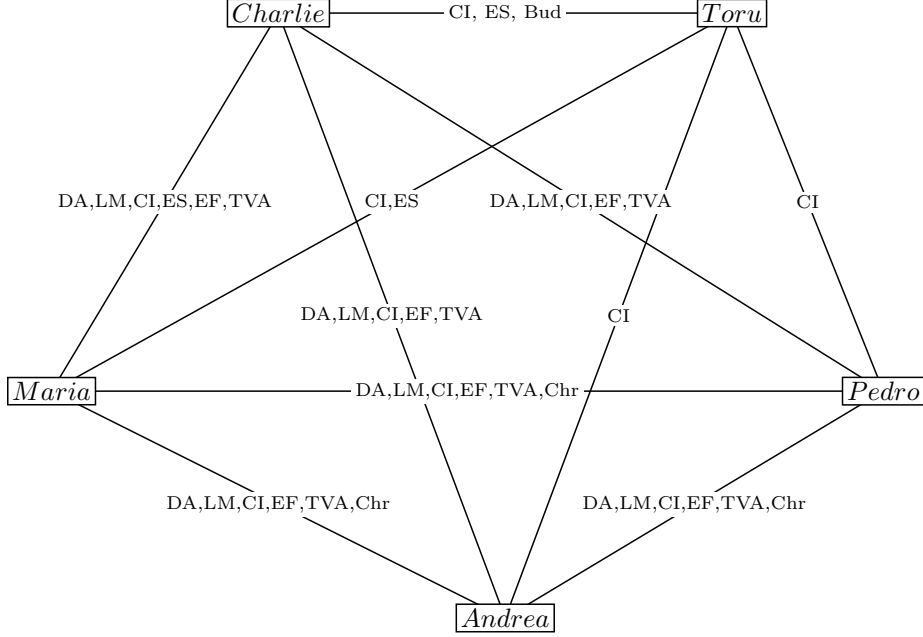Table 3.3: Traits of agents in Example 1.

Figure 3.1: The graph showing for which agents and traits the predicate *sharing* holds in Example 1. 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)*. The traits are abbreviated as in Table 3.2, i.e., *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*, *taking_vacation_in_August* as *TVA*, *Christianity* as *Chr*, *Buddhism* as *Bud*.

Definition 1 (sharing) *For each pair of agents $a_i, a_j \in Ag$ and for each trait $\tau \in \mathcal{T}$, a_i and a_j share the trait τ iff they both have such a trait:*

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

Example 1 (continued). In the example, we can write *sharing(Toru, Maria, eating_with_sticks)*, or *sharing(Pedro, Andrea, cappuccino_is_coffee)*, 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 3.1.

We can represent the restriction of *sharing* to specific agents and traits, like the set $\{(a_i, a_j, \tau) | \tau = cappuccino\_is\_coffee, \ a_i, a_j \in \{Charlie, Toru, Maria, Andrea, Pedro\}\}$ as in Figure 3.2. This figure shows how one trait, *cappuccino_is_coffee*, is shared by the set of agents. ⊘

Let us assume that if an agent a_i has a trait τ, the trait τ can be transmitted to another agent a_j and we use the predicate *transmitted(a_i, a_j, τ)* to represent this.

Axiom 1 $\forall a_i, a_j \in Ag, \ \forall \tau \in \mathcal{T} : \ transmitted(a_i, a_j, \tau) \rightarrow sharing(a_i, a_j, \tau)$

Note that the axiom does not necessarily imply that if $has(a_i, \tau)$ and $has(a_j, \tau)$ then *transmitted(a_i, a_j, τ)*. We represent *transmitted(a_i, a_j, τ)* in a graph by a directed edge from a_i to a_j labeled τ.
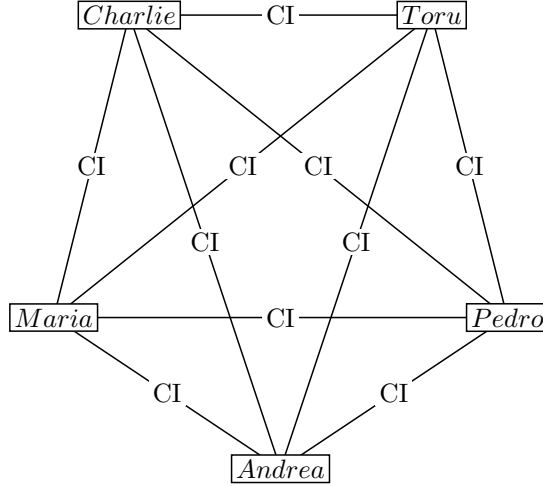
Figure 3.2:   The graph that shows for which agents the *sharing* predicate holds for the *cappuccino_is_coffee(CI)* trait in Example 1.
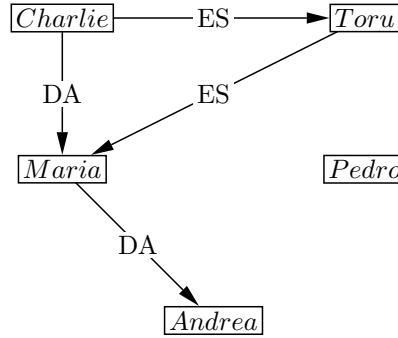
Figure 3.3: The graph representing the transmitted predicate. Each edge shows direction of the transmission of the trait in the label.

Example 1 (continued). Figure 3.3 shows the graph representing the *transmitted* predicate 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)*.

We can represent a restriction of *transmitted* to the set $\{(a_i, a_j, \tau)|a_i, a_j \in \{Charlie, Toru, Maria, Andrea, Pedro\}, \tau = Dante\_Alighieri\_wrote\_Divine\_Comedy\}$ as shown in Figure 3.4. ⊘

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

Definition 2 (weak sharing) *A set of traits T_G is weakly shared by a set of agents G 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.*
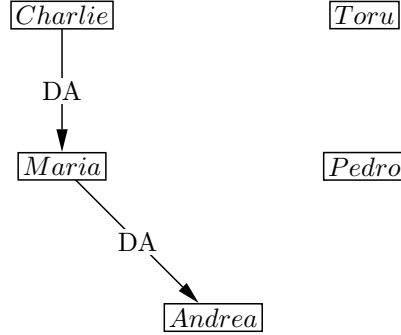
Figure 3.4: The transmission of the $Dante\_Alighieri\_wrote\_Divine\_Comedy(DA)$ trait.

Definition 3 (strong sharing) *A set of traits T_G is strongly shared by a set of agents G iff each trait $\tau \in T_G$ is shared by all pairs of agents $a_i, a_j \in G$.*

Example 1 (continued). Let us consider two sets of traits $T_G =\{cappuccino\_is\_coffee,$ $eating\_with\_sticks, Dante\_Alighieri\_wrote\_Divine\_Comedy\}, T'_G = \{cappuccino\_is\_coffee\}$, and the set $G = \{Charlie, Toru, Maria, Andrea, Pedro\}$. Using the *sharing* predicate represented in Figure 3.1, we can see that the *cappuccino_is_coffee* trait is shared by each pair of agents, so T'_G is strongly shared by G. T_G contains three traits that are shared by at least one pair of agents: e.g., *cappuccino_is_coffee*, *eating_with_sticks* shared by *Toru* and *Charlie*, *Dante_Alighieri_wrote_Divine_Comedy* shared by *Charlie* and *Andrea*. So, T_G is weakly shared by G. $\oslash$

Property 1 *Strong sharing implies weak sharing.*

Proof. Strong sharing of a set of traits T_G by a set of agents G means that for each $\tau \in T_G$ all pairs of agents $a_i, a_j \in G$ share τ. Thus, the condition for weak sharing, i.e. existence of one pair of agents $a_i, a_j \in G$, $a_i \neq a_j$ that share τ is fulfilled. $\square$

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 4 (culture of a set of agents) *A non-empty set of traits $T_G \subseteq \mathcal{T}$ is a culture of G iff*

- *the set T_G is weakly shared by G,*

- *for each trait $\tau \in T_G$ there exists an agent $a \in Ag$ that transmitted τ to another agent $a_j \in G$, i.e. transmitted(a, a_j, τ),*

- *for each agent $a \in G$ there exists a trait $\tau \in T_G$ such that has(a, τ).*

In other words, for a set of agents, a culture is defined as a set of transmitted traits weakly shared by agents, and each agent has at least one trait in the culture. Please, note that since the traits are transmitted not necessarily within the set, the transmission alone does not ensure sharing between the agents of G.

If T_G is also strongly shared then it is *a culture in a strong sense.*

Example 1 (continued). Considering $G = \{Charlie, Toru, Maria, Andrea, Pedro\}$ and the *transmitted* predicate as in Figure 3.3, $T'_G = \{cappuccino\_is\_coffee\}$ is not a culture because the *cappuccino_is_coffee* trait has not been transmitted. The same holds for $T_G = \{Dante\_Alighieri\_wrote\_Divine\_Comedy, cappuccino\_is\_coffee, eating\_with\_sticks\}$ because it contains the *cappuccino_is_coffee* trait. On the other hand, the set $T''_G = \{Dante\_Alighieri\_wrote\_Divine\_Comedy, eating\_with\_sticks\}$ is a culture since the traits *Dante_Alighieri_wrote_Divine_Comedy* and *eating_with_sticks* have been transmitted (from *Maria* to *Andrea* and from *Toru* to *Maria*, respectively), T''_G is weakly shared by G and each agent has at least one trait in T''_G (*Toru* has *eating_with_sticks*, and others have *Dante_Alighieri_wrote_Divine_Comedy*).

Let us consider a set $G' = \{Pedro, Maria\}$, and the set $T_{G'} = \{eating\_with\_sticks\}$. Although the trait *eating_with_sticks* has been transmitted to *Maria*, it is not a culture of G', because $T_{G'}$ is not a weakly shared by G'.

Taking $G'' = \{Charlie, Maria\}$, $T_{G''} = \{eating\_with\_sticks\}$ is a culture, because even that it has not been transmitted within the set, it has been transmitted to *Maria* from outside, it is shared by the set, and each agent has the *eating_with_sticks* trait.

Considering $G''' = \{Charlie, Maria, Andrea, Pedro\}$, $T_{G'''} = \{cappuccino\_is\_coffee, Dante\_Alighieri\_wrote\_Divine\_Comedy\}$ is not a culture, because the *cappuccino_is_coffee* trait has not been transmitted. The set $T'_{G'''} = \{Dante\_Alighieri\_wrote\_Divine\_Comedy\}$ is a strong culture since the *Dante_Alighieri_wrote_Divine_Comedy* trait has been transmitted (e.g. from *Charlie* to *Maria*), is owned by each agent, and the set $T'_{G'''}$ is strongly shared by G'''. $\oslash$

Property 2 *Given a set of agents $G \subseteq Ag$ and T_G, a culture of G, it is possible to find a non-empty set $G_0 \subseteq G$ and a non-empty set T_{G_0} such that T_{G_0} is a strong culture of G_0.*

Proof. If $|G| = 2$ then all traits that are weakly shared are also strongly shared and $T_{G_0} = T_G$ is a strong culture of $G_0 = G$. Otherwise, let us consider $G'_0 = \{a_1, a_2\}$, where a_1 and a_2 are two agents of G such that $T_{a_1} \cap T_{a_2} \cap T_G \neq \emptyset$. The existence of such a pair of agents is guaranteed, because $\forall a \in G \; \exists \tau \in T_a$ such that $\tau \in T_G$ and every $\tau \in T_G$ is weakly shared, so there are at least two agents that share it. For $T'_{G_0} = T_{a_1} \cap T_{a_2} \cap T_G \neq \emptyset$ being a culture of G'_0 it is necessary that $\forall \tau \in T'_{G_0} \; \exists a \in Ag, a_j \in G'_0$ such that $transmitted(a, a_j, \tau)$. Since T'_{G_0} is a subset of T_G and T_G is a culture of G, the following holds: $\forall \tau \in T'_{G_0} \; \exists a \in Ag, a_j \in G$ such that $transmitted(a, a_j, \tau)$. Let us take all $\tau \in T'_{G_0}$ such that the corresponding a_j is in G'_0. The set T_{G_0} composed of these traits is a strong culture of G'_0 because they are shared by both a_1 and a_2 and were transmitted to at least one of them. If it happens that T_{G_0} is empty, then let us take one trait τ_0 from T'_{G_0} and add the corresponding a_j to G_0. For the resulting set $G'_0 = \{a_1, a_2, a_j\}$, the set $T_{G_0} = \{\tau_0\}$ is a strong culture, because all agents in G_0 have this trait and it has been transmitted to $a_j \in G_0$. $\square$

Let us introduce sets $T^{tr}_{a_i}$ constructed in the following way: given an agent a_i, $T^{tr}_{a_i} = \{\tau : \tau \in T_{a_i}, \exists a \in Ag$ such that $transmitted(a, a_i, \tau)\}$. In other words, the set $T^{tr}_{a_i}$ contains the traits that were transmitted to a_i.

Property 3 *Given two agents $a_i, a_j \in Ag$, if the set of traits of a_i is a subset of or equal to the set of traits of a_j, it implies that $T^{tr}_{a_i}$ is a culture of the set of agents $G = \{a_i, a_j\}$.*

Proof. Let us show that $T_{a_i}^{tr}$ is a culture of G. The set $T_{a_i}^{tr}$ is a subset of T_{a_i} and a subset of T_{a_j}, so each $\tau \in T_{a_i}^{tr}$ is shared by the pair of agents a_i, a_j, so $T_{a_i}^{tr}$ is weakly shared by G. Moreover, it is easy to see that for both a_i and a_j there are traits, namely the whole set $T_{a_i}^{tr}$, which are in $T_{a_i}^{tr}$. From the definition of $T_{a_i}^{tr}$ it follows that for each $\tau \in T_{a_i}^{tr}$ exists $a \in Ag$ that transmitted τ to $a_i \in G$, i.e. the second condition of Definition 4 is also fulfilled, so $T_{a_i}^{tr}$ is a culture of G. $\square$

Property 4 *If for two agents $a_i, a_j \in Ag$, $T_G = T_{a_i}^{tr} \cap T_{a_j}^{tr}$ is not empty, this implies that T_G is a culture of the set of agents $G = \{a_i, a_j\}$.*

Proof. Let us show that $T_G = T_{a_i}^{tr} \cup T_{a_j}^{tr}$ is a culture of G. Since $T_{a_i}^{tr} \subseteq T_{a_i}$ and $T_{a_j}^{tr} \subseteq T_{a_j}$, $T_G \subseteq T_{a_i}$ and $T_G \subseteq T_{a_j}$, so T_G is weakly shared by G and for both a_i and a_j any trait $\tau \in T_G$ is also in T_{a_i} and T_{a_j}. From the definition of $T_{a_i}^{tr}$ and $T_{a_j}^{tr}$ it follows that for each $\tau \in T_G$ exists $a \in Ag$ that transmitted τ to some agent in G, i.e. the second condition of Definition 4 is also fulfilled, so $T_{a_i}^{tr}$ is a culture of G. $\square$

Property 5 *T_G is a culture of a set of agents G implies $T_G \subseteq \bigcup_{a_i \in G} T_{a_i}^{tr}$.*

Proof. By Definition 4, each $\tau \in T_G$ has been transmitted to some agent $a_j \in G$, so it is in $T_{a_j}^{tr}$ and all $\tau \in T_G$ constitute a subset of $\bigcup_{a_i \in G} T_{a_i}^{tr}$. $\square$

3.3 Dynamics of culture

In the previous section, we defined culture of a set of agents, highlighting some important properties a set of traits must possess to be a culture of the set of agents. Those definitions considered an agent as a constant set of traits. However, we can hardly imagine that the set of traits of an agent remain constant over time. Therefore, in this section, we introduce the notion of state and use it to model changes in the set of traits of an agent and consequently, changes in culture.

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 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 $T_a(s) = \{\tau_i\} \subseteq \mathcal{T}$ and we use the predicate $has(a, \tau_i, s)$ to represent the fact that the agent a has the trait $\tau_i \in T_a(s)$ in the state s.

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" [123]. We define the set of all behaviors $\mathcal{B} \subseteq \mathcal{T}$ and the function $perform$ in $Ag \times \mathcal{B} \times S \rightarrow S$. 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 $\mathcal{B} \subseteq \mathcal{T}$ we do not include innate behaviors, such as blinking when air is puffed in someone's eye.

At this point we would like to 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 Example 1, when someone needs to know whether agents are working, it is possible to consider behaviors *working* and *eating*, or, even, *working* and *not_working*. However, if someone would like to have a closer look at eating habits of the group, it is necessary to introduce finer granularity of the *eating* behavior, e.g. by considering *eating_with_sticks* and *eating_with_fork* behaviors.

Let us introduce a specific behavior $do\_nothing \in \mathcal{B}$ that means that an agent does not perform any other behavior in $\mathcal{B}$ and assume that each agent has this behavior in every state. For simplifying notation we usually omit the behavior $do\_nothing$ from the description of the states of the agents. Performing $do\_nothing$ does not change the state of the world. We assume that it is not possible to perform more than one behavior concurrently in the world, with the exception of $do\_nothing$, and if an agent performs a behavior, all the other agents perform $do\_nothing$. Finally, we assume that the states are ordered, we define recursively the order "is before" and the corresponding predicate $is\_before(s_u, s_v)$ in the following way:

Definition 5 (is_before) $is\_before(s_u, s_v) \leftrightarrow \exists a \in Ag,\ \tau \in \mathcal{B},\ s \in S$ *such that* $s = perform(a, \tau, s_u) \land (s = s_v \lor is\_before(s, s_v))$.

Analogously, "is_after" is defined as:

Definition 6 (is_after) $is\_after(s_v, s_u) \leftrightarrow is\_before(s_u, s_v)$

We also state the following axiom:

Axiom 2 *For all agents* $a \in Ag$, *for all behaviors* $\tau \in \mathcal{B}$ *and for all states* $s_u, s_v \in S$

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

In the perspective of states we define *sharing* and *transmitted* predicates.

Definition 7 (sharing) *For each pair of agents* $a_i, a_j \in Ag$, *for each trait* $\tau \in \mathcal{T}$, *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:

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

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

set	traits
$\mathcal{T}$	*Dante_Alighieri_wrote_Divine_Comedy*, *Meiji_era_was_in_1868_1912*, *latte_macchiato_is_coffee*, *cappuccino_is_coffee*, *eating_with_sticks*, *eating_with_fork*, *taking_vacation_in_August*, *taking_vacation_in_May*, *never_put_mayonnaise_on_pizza*, *never_open_umbrella_inside_building*, *take_only_week_of_vacation_per_year*, *never_drink_cappuccino_after_lunch*, *Christianity*, *Buddhism*, **telling**, **memorizing**
$T_{Charlie}(s_1)$	*Dante_Alighieri_wrote_Divine_Comedy*, *latte_macchiato_is_coffee*, *cappuccino_is_coffee*, *eating_with_sticks*, *eating_with_fork*, *taking_vacation_in_August*, *never_put_mayonnaise_on_pizza*, *Buddhism*, **telling**
$T_{Pedro}(s_1)$	*Dante_Alighieri_wrote_Divine_Comedy*, *latte_macchiato_is_coffee*, *cappuccino_is_coffee*, *eating_with_fork*, *taking_vacation_in_August*, *never_drink_cappuccino_after_lunch*, *Christianity*
$T_{Toru}(s_1)$	*Meiji_era_was_in_1868_1912*, *cappuccino_is_coffee*, *eating_with_sticks*, *taking_vacation_in_May*, *Buddhism*, **memorizing**
$T_{Maria}(s_1)$	*Dante_Alighieri_wrote_Divine_Comedy*, *latte_macchiato_is_coffee*, *cappuccino_is_coffee*, *eating_with_sticks*, *eating_with_fork*, *taking_vacation_in_August*, *Christianity*
$T_{Andrea}(s_1)$	*Dante_Alighieri_wrote_Divine_Comedy*, *latte_macchiato_is_coffee*, *cappuccino_is_coffee*, *eating_with_fork*, *taking_vacation_in_August*, *Christianity*

Table 3.4: Traits of agents in Example 2. Differences with Example 1 are put in bold.

Axiom 3 *For all agents $a \in Ag$, traits $\tau \in \mathcal{T}$, and states $s \in S$:*

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

Example 2. In this section, we consider the following example that is an extension of Example 1 in Section 3.2 with states. Again, we 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*. In these settings, the set of agents is $Ag = \{Charlie, Pedro, Toru, Maria, Andrea\}$, the set of all traits $\mathcal{T}$ is as shown in Table 3.4. This table also lists the sets of traits of agents in the initial state s_1.

The predicate *sharing* in the state s_1 is identical to the predicate *sharing* in Example 1. Thus, when considering only the state s_1 the predicate *sharing* is as in Figure 3.1. ⊘

Definition 8 (transmitted) *For each pair of agents $a_i, a_j \in Ag$, $a_i \neq a_j$, for each trait $\tau \in \mathcal{T}$, 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:*

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

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, as shown by the following property:

Figure 3.5: The graph that shows for which agents the *transmitted* predicate holds in the state s_3 in Example 2. Changes with respect to state s_1 are put in bold.

Property 6 *For all pairs of agents $a_i, a_j \in Ag$, for all traits $\tau \in \mathcal{T}$, and for all states $s_v \in S$*

$$sharing(a_i, a_j, \tau, s_v) \rightarrow (\forall s : is\_after(s, s_v)\ sharing(a_i, a_j, \tau, s))$$

Proof. The proof follows from Axiom 3. $\square$

Property 7 *For all pairs of agents $a_i, a_j \in Ag$, for all traits $\tau \in \mathcal{T}$, and for all states $s_v \in S$*

$$transmitted(a_i, a_j, \tau, s_v) \rightarrow (\forall s : is\_after(s, s_v)\ transmitted(a_i, a_j, \tau, s))$$

Proof. The proof follows from Definition 8, as s_v we just take the same s_u whose existence is required for s. $\square$

Example 2 (continued). The predicate *transmitted* in the state s_1 is identical to the predicate *transmitted* of Example 1 and so it is the same as in Figure 3.3.

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 Figure 3.3 and *transmitted* in the state s_3 is as depicted in Figure 3.5. 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: $T_{Toru}(s_3) = \{Meiji\_era\_was\_in\_1868\_1912,$ *Dante_Alighieri_wrote_Divine_Comedy, cappuccino_is_coffee, eating_with_sticks, taking_vacation_in_May, Buddhism, memorizing*}.

Obviously, the transmission has an impact on sharing and the *sharing* predicate in the state s_3 is as depicted in Figure 3.6, with the edges between *Toru* and *Charlie*, *Maria*, *Andrea*, *Pedro* added. $\oslash$

It is easy to see if we fix a state $s \in S$ the predicates *transmitted* and *sharing* correspond to the predicates *transmitted* and *sharing* defined in Section 3.2. Given that, it is possible to define weakly (strongly) shared set of traits and a culture of a set of agents in a state:
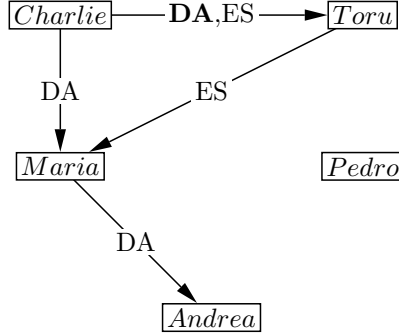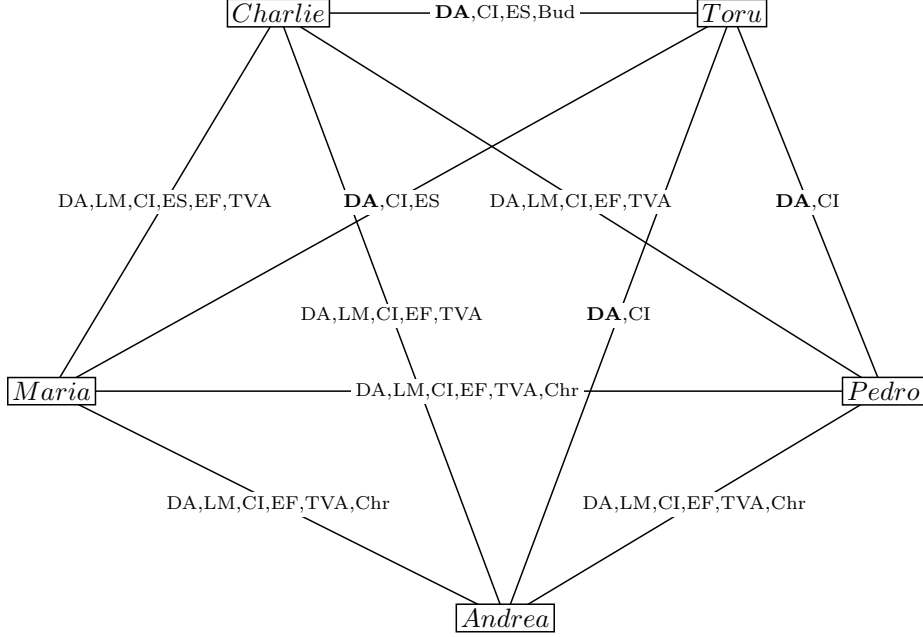
Figure 3.6: The graph that shows for which agents the *sharing* predicate holds in the state s_3 in Example 2. Changes with respect to state s_1 are put in bold.

Definition 9 (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 10 (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 if it is a subset of the union (intersection) of traits shared by pairs of agents of G in the state s.

Example 2 (continued). Let us consider the set of agents $G = \{Charlie, Toru, Maria, Andrea, Pedro\}$. Analyzing the *sharing* predicate in the state s_1 (Figure 3.1) 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.

Analogously, the set $T''_G = \{eating\_with\_sticks, Dante\_Alighieri\_wrote\_Divine\_Comedy, cappuccino\_is\_coffee\}$ is weakly shared by G in the state s_3, and the set $T'''_G = \{Dante\_Alighieri\_wrote\_Divine\_Comedy, cappuccino\_is\_coffee\}$ is strongly shared by the set G in the state s_3. $\oslash$

Definition 11 (culture of a set of agents) *A non-empty set of traits $T_G \subseteq \mathcal{T}$ is a culture of G in a state s iff*

- *the set T_G is weakly shared by G in the state s,*

- *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. $transmitted(a, a_j, \tau, s)$,*

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

From this definition 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.

If the set of traits T_G is strongly shared then it is *a culture in a strong sense*.

Example 2 (continued). 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.

Although 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 the $Dante\_Alighieri\_wrote\_Divine\_Comedy$ trait in those states. $\oslash$

The following proposition outlines some restrictions on how culture can change between states, namely it shows that culture is monotonic.

Proposition 1 (monotonicity of culture) *If a non-empty set of traits T_G is a culture of a set of agents G in a state s_v, then T_G is a culture of G also in any state s after s_v.*

Proof. Using Property 6 it is easy to see that if T_G is weakly shared by G in the state s_v, it is also weakly shared in any subsequent state s. The condition that for each trait $\tau \in T_G$ exists an agent $a \in Ag$ and an agent $a_j \in G$ such that $transmitted(a, a_j, \tau, s)$ is fulfilled using Property 7. Finally, using Axiom 3 we also have that for each agent $a \in G$ in the state s there exists a trait $\tau \in T_G$ such that $has(a, \tau, s)$, because $has(a, \tau, s_v)$. So, T_G is a culture of G in the state s. $\square$

In real world, the traits of a culture can be lost for two reasons: (1) agents can lose traits, (2) agents can die, move to another group, etc. As we stated in Axiom 3, in our model, agents do not lose traits. However, our model, and the proposition about monotonicity of culture support the case when agents disappear from the group.

Definition 12 (maximal culture of a group) *A non-empty set of traits T_G^{max} is the maximal culture of a set of agents G in the state s iff T_G^{max} is the union of all cultures T_G of G in the state s.*

In other words, the maximal culture of a set of agents in some state is the union of all possible cultures of the set in this state. Since it is the union of all cultures, it is not

possible to add any trait to T_G^{max} and still obtain a culture of G. In the following, we refer to maximal culture of a set of agents as "the culture of a set".

Note, that similar definition could have been provided in Section 3.2, but we are using just the definition of maximal culture with states.

Definition 13 (evolution of culture) *A sequence of sets of traits* $\{T_G^{(1)}, \ldots, T_G^{(i)}\}$ *is an evolution of culture of G iff:*

- *exists a sequence of states* $\{s_1, \ldots, s_i\}$, *such that* $T_G^{(k)}$ *is a culture of G in the state s_k for all k, $1 \leq k \leq i$,*

- *for each k, $1 \leq k \leq i - 1$ holds is_after(s_{k+1}, s_k).*

In other words, a sequence of sets of traits is an evolution of culture if each set of traits in the sequence is a culture of G in some state and the states are ordered in the same way as the sets of traits. We denote evolution of culture as $\{T_G\}$.

3.4 Problems involving culture

The formalisms that we presented in previous sections can be used to express a range of practical problems involving culture. In this section, we present classes of such problems, and each instance of a class can be encountered in a broad range of applications. The main purpose of this section is to provide an abstract classification of problems that involve culture, so concrete examples of problems involving culture can be mapped to this classification.

We classify the problems that involve culture based on their inputs and outputs, as shown in Table 3.5. Based on the problem outputs we introduce the following broad classes of problems: *discover*, which includes problems dealing with finding either a set of agents or a culture or an evolution of a culture; *achieve_state*, which contains problems where a state of the world satisfying some conditions must be achieved; *evaluate*, which contains problems dealing with evaluating culture with a range of metrics[2].

The *discover* class is further divided on subclasses that depend on the outputs. We assume that each class of problems can have only one output reflected in the name of the class: *discover_set* problems have a set of agents as the output; *discover_culture* problems have a culture as the output; *discover_evolution* problems have an evolution of cultures of some set of agents as the output. The output of *achieve_state* problems is a sequence of states[3]; and the output of *evaluate* problems is a set of values of some metrics calculated on the inputs.

We decided to put *achieve_state* as a separate class rather than as a subset of *discover* class because this class of problems includes problems such as "how the world should evolve in order to..." rather than "when and how did it happen that..." that occur in the *discover* class. In the *evaluate* class we do not consider problems that evaluate a set of agents or states because in this thesis we are interested in culture.

---

[2]We elaborate more on the metrics for culture in Section 3.5.

[3]Please note that we treat one state in the output as a particular case of the sequence of states that has the length one, while for culture we have separate outputs - a culture or a culture in a sequence of states.

N	subclass	G	T_G	states	output	example
Discover						
1	discover_set_of_agents	-	given	-	G	Find a set of agents that have the given culture
2	discover_set_of_agents	given	given	-	G	Given a set of agents and its culture find another set of agents that have this culture
3	discover_set_of_agents	-	given	given	G	Find a set of agents that have the given culture in the given state
4	discover_set_of_agents	-	given evolution	-	G	Find a set of agents that have the culture as specified by the given evolution
5	discover_set_of_agents	-	given evolution	given	G	Find a set of agents that have the given evolution of culture in the given states
6	discover_set_of_agents	given	given	given	G	Given a set of agents and its culture in the given state, find another set of agents that have this culture in this state
7	discover_set_of_agents	given	given evolution	-	G	Given a set of agents and the evolution of its culture , find another set of agents that have the culture as specified by the evolution
8	discover_set_of_agents	given	given evolution	given	G	Given a set of agents and the evolution of its culture, find another set of agents that have such evolution of culture in these states
9	discover_culture	given	-	-	T_G	Find a culture of the given set of agents
10	discover_culture	given	given	-	T_G	Find a culture of the given set of agents such that this culture includes the given culture
11	discover_culture	given	-	given	T_G	Find a culture of the given set of agents in the given state
12	discover_culture	given	given	given	T_G	Given a culture of the given set of agents in some state find a culture of this set in another state
13	discover_culture	given	given evolution	-	T_G	Given a set of agents and the evolution of its culture in unknown states, find a culture of the set that is present in every element of the evolution
14	discover_culture	given	given evolution	given	T_G	Given a set of agents and the evolution of its culture in a sequence of states, find a culture of the set of agents in the next state
15	discover_evolution	given	-	-	$\{T_G\}$	Find how a culture of the given set of agents can evolve
16	discover_evolution	given	given	-	$\{T_G\}$	Given a culture of the set of agents find how it can evolve
17	discover_evolution	given	given evolution	-	$\{T_G\}$	Given the evolution of a culture of another set of agents in unknown states find how the culture of the given set of agents evolves
18	discover_evolution	given	-	given	$\{T_G\}$	Given a set of agents and a sequence of states find how a culture of the set evolves in these states
19	discover_evolution	given	given	given	$\{T_G\}$	Given a set of agents, a sequence of states and a culture of the set in one of the states find how this culture of the set evolves in the other states
20	discover_evolution	given	given evolution	given	$\{T_G\}$	Given the evolution of a culture of a set of agents in some states find how a culture of the set evolves in the other states specified by the input
Achieve_state						
21	achieve_state	given	given	-	s	Given a culture and a set of agents find a state in which the culture is a culture of the set
22	achieve_state	given	given evolution	-	$\{s\}$	Given a culture evolution and a set of agents find a sequence of states in which the elements of the evolution are cultures of the set
23	achieve_state	given	given	given	s	Given a culture of a set of agents in some state preserve this culture as a culture of the set also in the next state
24	achieve_state	given	given evolution	given	s	Given the evolution of a culture of a set of agents preserve this evolution of culture in the other given states
Evaluate						
25	evaluate	given	given	-	values	Evaluate the given culture of the given set of agents
26	evaluate	given	given	given	values	Evaluate the given culture of the given set of agents in the given state
27	evaluate	given	given evolution	-	values	Evaluate the given evolution of culture of the given set of agents in the unknown states
28	evaluate	given	given evolution	given	values	Evaluate the given evolution of culture of the given set of agents in the given states

Table 3.5: Classification of problems involving culture. G denotes a set of agents, T_G denotes a culture of a set of agents G, $\{T_G\}$ denotes an evolution of culture, s denotes states, $\{s\}$ denotes a sequence of states, and *values* stand for the values of the different metrics calculated on a culture or an evolution of a culture.

The following inputs are considered for the problems: a set of agents; a set of traits, i.e. a culture of the set; evolution, i.e. a culture of the set in a sequence of states; states, one or a sequence. The set of agents can be given or not, the culture can be given, given as evolution, or not given, and the states can be given or not given. For each class of problems we list the possible combination of inputs and give an example of the problem. Please note that each class of problems can contain an infinite number of concrete problems based on the problem inputs and outputs, and thus example refers only to one instance of the problem. We do not specify how evolution of a culture of a set of agents should be considered: as a sequence of maximal cultures, as a sequence of cultures that include some given set of traits, etc. We will illustrate some of the possible variations of semantics in examples of problem definitions when describing problem classes.

When determining the possible combinations of inputs we used the following assumptions that can be considered as constraints on inputs:

- Each problem can have only one output as specified previously: a set of agents, a culture, an evolution of culture, a state, values of metrics.

- The set of agents is either given or it is the output. This is due to the fact that culture is impossible without a set of agents.

- The culture is either given or it must be the output. This is natural, taking into account the fact that we are considering problems involving culture.

- We do not consider cases when the set of agents evolves over time, again because we are more focused on culture.

- For the same reason we are considering only metrics on culture, not metrics on agents or on states.

- We are aware of the possibility of specifying given culture or set of agents by means of their intensional as opposed to extensional definition, i.e. giving listing all the properties required for belonging to the set rather than enumerating all the members of the set, but addressing intensional definitions of culture or a set of agents is out of the scope of this thesis.

Table 3.5 shows the list of problem classes divided in the three main classes. For the specified inputs, outputs, and constraints, the classification is complete. Only some of the problem classes — those numbered 1, 2, 9, 10, 25 — can be expressed in the terms of the formalism presented in Section 3.2. This is possible because these classes of problems do not involve states. Even though every problem expressed in this formalism can be expressed in the the formalism presented in Section 3.3, e.g. given some state, the contrary does not hold. Considering the specified values of the inputs and the constraints, the list of problems is complete.

Let us show that it is possible to express the presented problems using our formalism. To do this we represent each example in the table using terms in our formalism:

1. Given a set of traits T_G, find a set of agents G such that T_G is a culture of G.

2. Given a set of agents G and a culture T_G of G, find another set of agents $G' \neq G$ such that T_G is a culture of G'.

3. Given a set of traits T_G and a state s, find a set of agents G such that T_G is a culture of G in the state s.

4. Given a sequence of sets of traits $\{T_G\}$, find a set of agents G such that the evolution of the culture of G is as specified by the sequence.

5. Given a sequence of sets of traits $\{T_G\}$ in a sequence of states $\{s\}$, find a set of agents G such that for each s from the sequence of states the corresponding $T_G(s)$ is a culture of G in the state s.

6. Given a set of agents G, a state s, and a culture T_G of G in the state s, find another set of agents $G' \neq G$ such that T_G is a culture of G' in the state s. *Note that in this example we can additionally ask for $G' \subseteq G$, $G \subseteq G'$, etc.*

7. Given a set of agents G and an evolution a culture of G, $\{T_G\}$, find another set of agents $G' \neq G$ such that the evolution of the culture of G' is equal to $\{T_G\}$. *Note that we can ask that the evolution of the culture of G' contains $\{T_G\}$.*

8. Given a set of agents G and an evolution a culture of G, $\{T_G\}$, in a sequence of states $\{s\}$, find another set of agents $G' \neq G$ such that for each s from the sequence of states the corresponding $T_G(s)$ is a culture of G' in the state s.

9. Given a set of agents G, find a set of traits T_G such that it is a culture of G. *Note that the output is not uniquely identified, but we can require to find the maximal culture (uniquely identified) of the set.*

10. Given a set of agents G and a culture T_G of the set G, find a set of traits T'_G such that it is a culture of G and $T_G \subseteq T'_G$.

11. Given a set of agents G and a state s, find a set of traits T_G such that it is a culture of G in the state s.

12. Given a set of agents G, states s and s', and a culture T_G of the set G in s, find a set of traits T'_G such that it is a culture of G in the state s'.

13. Given a set of agents G and an evolution of a culture of G, $\{T_G\}$, find a set of traits T'_G such that it is a subset of each element in the sequence $\{T_G\}$, i.e., find culture that preserves over time.

14. Given a set of agents G, and an evolution of a culture of G, $\{T_G\}$, in a sequence of states $\{s\}$, find a set of traits T'_G such that it is a culture of G in the next state.

15. Given a set of agents G, find an evolution of a culture of G, $\{T_G\}$.

16. Given a set of agents G and a culture T_G of the set G, find the evolution of T_G, $\{T_G\}$.

17. Given sets of agents G and G', and an evolution of a culture T_G of the set G, $\{T_G\}$, find an evolution of a culture of the set G', $\{T'_G\}$.

18. Given a set of agents G and a sequence of states $\{s\}$, find an evolution of a culture of G, $\{T_G\}$, in this sequence of states.

19. Given a set of agents G, a sequence of states $\{s\}$, and a culture T_G of G in one of the states, s, find the evolution of T_G, $\{T_G\}$, in the other states from the sequence.

20. Given a set of agents G, a sequence of states $\{s\}$, and an evolution of culture T_G of G in some of the states, $\{T_G\}$, find the evolution of T_G, $\{T_G\}$, in the other states from the sequence.

21. Given a set of agents G and a culture T_G of the set G, find a state s such that T_G is a culture of G in the state s.

22. Given a set of agents G and a sequence of sets of traits $\{T_G\}$, find a sequence of states $\{s\}$ such that the elements of $\{T_G\}$ are cultures of G in the corresponding states.

23. Given a set of agents G, a state s, and a culture T_G of the set G in s, preserve T_G as a culture of G also in the next state s'.

24. Given a set of agents G, a sequence of states $\{s\}$, and an evolution of culture T_G of the set G in the subset of the sequence of states, $\{T_G\}$, preserve the same evolution of culture also in the other states of the sequence.

25. Given a culture T_G of a set of agents G, calculate values of desired metrics.

26. Given a set of agents G, a state s, and a culture T_G of the set G in the state s, calculate values of desired metrics.

27. Given an evolution of a culture $\{T_G\}$ of a set of agents G, calculate values of desired metrics.

28. Given an evolution of a culture $\{T_G\}$ of a set of agents G in a sequence of states $\{s\}$, calculate values of desired metrics.

The classes of problems presented in this section occur in a wide range of applications. For instance, anthropological research is included into the *discover_culture* and *discover_evolution classes*, while the *discover_set_of_agents* class includes problems of personnel hiring in the organizational settings [69, p. 21]. O'Reilly [109] shows that to maintain the strong culture in an organization, one might select members based on cultural criteria. In our terms, this problem can be formulated as the need to preserve the culture in the successive states, and it falls into the *achieve_state* class (problem class 23 in the classification). As a particular instance of problems from the *evaluate* class, we can mention the study of why some cultures endure longer than other [31].

3.5 Measures for comparison of cultures

In this section, we present some measures for characterizing a culture of a set of agents in different socio-cultural settings and for comparing cultures of different sets. This list is not exhaustive, rather, it contains some initial measures, and further extension of this list is a subject of future research.

3.5.1   Measuring culture as a snapshot

Culture

Let us start from simple measures such as *presence of a specific trait in a culture.* We use an indicator function $I_{has}(\tau, T)$ to say that the trait τ is present in the culture T:

$$I_{has}(\tau, T) = \begin{cases} 1, & \text{if } \tau \in T \\ 0, & \text{otherwise} \end{cases} \tag{3.1}$$

Another example of a simple measure of a culture could be *the number of traits in the culture*, defined as $|T|$, i.e. the dimension of the set of traits T.

Culture of a group

A culture of a group is a product of the individuals belonging to the group. However, different groups can share cultures to some extent. To measure such degree of sharing we adapt the notion of cultural homogeneity introduced by Carley in [31]. Culture in that paper is defined as the distribution of information (ideas, beliefs, concepts, technical knowledge, etc.) across population. In our settings, given a set of agents G and a culture T_G of G, the cultural homogeneity is measured by the percentage of possible dyadically shared traits that actually are shared. A trait τ is shared by a dyad if $sharing(a_i, a_j, \tau)$. The number of possible dyadically shared traits is $\binom{N}{2} \times K$, where N is the number of agents in the set, $|G|$; $K = |T_G|$ is the number of traits in the culture T_G. Thus, *cultural homogeneity* is measured as

$$CH(G, T_G) = \frac{\sum_{i=1}^{N} \sum_{j=i+1}^{N} \sum_{k=1}^{K} I_{sharing}(a_i, a_j, \tau_k)}{\binom{N}{2} \times K} \times 100\%. \tag{3.2}$$

In this formula, $G = \{a_i\}$, $1 \le i \le N$, $T_G = \{\tau_k\}$, $1 \le k \le K$, and the indicator function $I_{sharing}$ is defined as follows:

$$I_{sharing}(a_i, a_j, \tau_k) = \begin{cases} 1, & \text{if } sharing(a_i, a_j, \tau_k) \\ 0, & \text{otherwise.} \end{cases}$$

It is easy to note that the cultural homogeneity takes into account only traits present in the culture, and it does not matter what traits agents of G have besides those contained in the culture T_G. To take the traits that are not a part of culture into account, we introduce the notion of group homogeneity. To do this, we need to consider the set of all traits of the group $\bar{T}_G = \cup_{i=1}^{N} a_i$, $\bar{K} = |\bar{T}_G|$. Thus, *group homogeneity* of the group G is measured as

$$GH(G) = \frac{\sum_{i=1}^{N} \sum_{j=i+1}^{N} \sum_{k=1}^{\bar{K}} I_{sharing}(a_i, a_j, \tau_k)}{\binom{N}{2} \times \bar{K}} \times 100\%, \tag{3.3}$$

where τ_k, $1 \le k \le \bar{K}$ are from the set $\bar{T}_G$ and the other terms are defined in Equation 3.2.

A culture of an individual and a culture of a group

To compare a culture of an individual a and a culture of a group G we introduce the following measures:

- *Common culture (culture overlap)* is the set of traits that is present in both cultures: $CC(T_a, T_G) = T_a \cap T_G$.

- *Culture similarity* is the degree to which two cultures are similar, i.e. how much they have in common: $CS(T_a, T_G) = \frac{|T_a \cap T_G|}{|T_a \cup T_G|} \times 100\%$.

- *Culture fit* is the degree to which one culture fits the other culture: $CF(T_a, T_G) = \frac{|T_a \cap T_G|}{|T_G|} \times 100\%$.
  Note that this measure is not symmetric.

Note that it is possible to extend the notion of culture similarity further if we assume there is a domain-specific function for calculating similarity between traits, i.e. for each pair of traits τ_1, τ_2 we know the value of $sim(\tau_1, \tau_2)$. Culture similarity can then be defined as $CC(T_a, T_G) = \frac{\sum_{i=1}^{|T_a|} \sum_{j=1}^{|T_G|} sim(\tau_i^a, \tau_j^G)}{|T_a| \times |T_G|} \times 100\%$

This will allow for considering the degree of similarity between different traits, e.g., specifying that trait *eating_with_sticks* is more similar to *eating_with_fork* than to *telling*.

A culture of a group and a culture of another group

In order to compare cultures of two sets of agents we can straightforwardly replace the culture of an individual with a culture of another group in the formulas above, thus introducing the following measures:

- Common culture (culture overlap) is the set of traits that is present in both cultures: $CC(T_{G_1}, T_{G_2}) = T_{G_1} \cap T_{G_2}$.

- Culture similarity is the degree to which two cultures are similar, i.e. how much they have in common: $CS(T_{G_1}, T_{G_2}) = \frac{|T_{G_1} \cap T_{G_2}|}{|T_{G_1} \cup T_{G_2}|} \times 100\%$.

- Culture fit is the degree to which one culture fits the other culture: $CF(T_{G_1}, T_{G_2}) = \frac{|T_{G_1} \cap T_{G_2}|}{|T_{G_2}|} \times 100\%$.
  Note that this measure is not symmetric.

3.5.2 Measuring culture evolution

We can also extend some of the measures to deal with culture in different states. For instance, we can see the *presence of a specific trait in a culture in a state*:

$$I_{has}(\tau, T, s) = \begin{cases} 1, & \text{if } \tau \in T(s) \\ 0, & \text{otherwise} \end{cases} \tag{3.4}$$

measure	meaning
$I_{has}(\tau, T)$	shows if the trait τ is present in the culture T
$|T|$	the number of elements in the culture T
$CH(G, T_G)$	cultural homogeneity of G, i.e. how widely the culture T_G is shared within the group G
$GH(G)$	group homogeneity, i.e. how similar are the sets of traits of agents of G
$CC(T_a, T_G)$ $CC(T_{G_1}, T_{G_2})$	common culture, i.e. the set of traits contained in the culture of an agent a (a group G_1) and in the culture T_G (T_{G_1})
$CS(T_a, T_G)$ $CS(T_{G_1}, T_{G_2})$	culture similarity, i.e. how much two cultures have in common
$CF(T_a, T_G)$ $CF(T_{G_1}, T_{G_2})$	culture fit, i.e. the degree to which the culture of a (G_1) fits the culture T_G (T_{G_2})

Table 3.6: Measures of culture as a snapshot.

Extending other formulas to deal with states is rather straightforward. For instance, given a set of agents G and a culture T_G of G, the cultural homogeneity in a state s is measured by the percentage of possible dyadically shared traits that actually are shared in this state and calculated as follows:

$$CH(G, T_G, s) = \frac{\sum_{i=1}^{N} \sum_{j=i+1}^{N} \sum_{k=1}^{K} I_{sharing}(a_i, a_j, \tau_k, s)}{\binom{N}{2} \times K} \times 100\%. \qquad (3.5)$$

In this formula, $G = \{a_i\}$, $1 \leq i \leq N$, $T_G = \{\tau_k\}$, $1 \leq k \leq K$, and the indicator function $I_{sharing}$ is defined as follows:

$$I_{sharing}(a_i, a_j, \tau_k, s) = \begin{cases} 1 \text{ , if } sharing(a_i, a_j, \tau_k, s) \\ 0 \text{ , otherwise.} \end{cases}$$

For the measures of culture of two groups, we can also take T_{G_1} as a culture in one state and T_{G_2} as a culture of the same group in another state and see how culture of the same group changes between states and measure spread of some specific trait within a set of agents.

3.5.3 Example

Let us see how the described measures apply to Example 1 from Section 3.2, summarized in Table 3.3. Considering a set of agents $G = \{Charlie, Toru, Andrea, Maria, Pedro\}$, and a culture $T_G = \{Dante\_Alighieri\_wrote\_Divine\_Comedy(DA), eating\_with\_sticks(ES)\}$:

- $I_{has}(eating\_with\_sticks, T_G) = 1$,

- $I_{has}(eating\_with\_fork, T_G) = 0$,

- $|T_G| = 2$.

To calculate the cultural homogeneity of G we need to calculate the number of traits in the culture T_G: $K = 2$ and the number of agents in the set G: $N = 5$. With these

parameters, $CH(G, T_G)$ is calculated as follows:

$$CH(G, T_G) = \frac{\sum_{i=1}^{5} \sum_{j=i+1}^{5} \sum_{k=1}^{2} I_{sharing}(a_i, a_j, \tau_k)}{\binom{5}{2} \times 2} \times 100\% =$$

$$= \frac{\sum_{i=1}^{5} \sum_{j=i+1}^{5} (I_{sharing}(a_i, a_j, DA) + I_{sharing}(a_i, a_j, ES))}{10 \times 2} \times 100\% =$$

$$\frac{6+3}{20} \times 100\% = 45\%.$$

Proceeding with calculations we get:

- $GH(G) = \frac{41}{120} \times 100\% = 34.17\%$,

- $CC(Pedro, T_G) = \{Dante\_Alighieri\_wrote\_Divine\_Comedy\}$,

- $CS(Pedro, T_G) = 0.125$,

- $CF(Pedro, T_G) = 0.5$,

- $CF(T_G, Pedro) = 0.143$.

3.6 A case study

In this section, we provide a case study that shows how the material presented in this chapter can be applied in the Web 2.0 domain. We first describe the scenario and then show how it can be addressed with our approach.

3.6.1 Scenario description

Let us consider activities related to bibliography management in `CiteULike.org`, a free online service to organize someone's collection of academic papers. Users of CiteULike are mainly scientists and there are groups dedicated to specific interests. The site allows people to add papers in their personal collections or to the collections of the groups users belong to and to tag those papers. It is also possible to search for the papers using keywords or browse the papers with a specific tag.

Let us suppose that Michael, a user of CiteULike, has some papers about recommendation systems in his bibliography and has tagged them as shown in Table 3.7[4]. He discovers that there are groups on CiteULike and that there are at least three groups that seem relevant to his research interests: GroupA, GroupB, and GroupC. In the group bibliography, each group has a list of papers tagged as shown in Table 3.7. Michael would like to join some group, but he does not have much time to read group feeds, so he would like

---

[4]Of course, we present a simplified example here, real users and groups on CiteULike have much more papers in their bibliographies.

Michael		
paperID	**paper**	**tags**
PolyLens	PolyLens: a recommender system for groups of users	recommendation, collaborative filtering
TrustInRS	Trust in recommender systems	trust, recommendation
GroupLens	GroupLens: An Open Architecture for Collaborative Filtering of Netnews	collaborative filtering, grouplens
RefWeb	Referral Web: Combining Social Networks and Collaborative Filtering	collaborative filtering, trust
TrustCF	Trust-Aware Collaborative Filtering for Recommender Systems	trust, recommendation
Group A		
EComRec	E-Commerce Recommendation Applications	collaborative filtering, ecommerce, recommender
TechLens	Enhancing digital libraries with TechLens+	recommender, academic reference
GetToKnow	Getting to know you: learning new user preferences in recommender systems	collaborative filtering, recommender
GroupLens	Group Lens: An open architecture for collaborative filtering of netnews	collaborative filtering, recommender
PolyLens	PolyLens: a recommender system for groups of users	recommendation, collaborative filtering
Group B		
TechLens	Enhancing Digital Libraries with TechLens+	collaborative filtering, content based filtering, papers, recommender systems
Citations	On the Recommending of Citations for Research Papers	citations, collaborative filtering, personalization, recommender systems
Scouts	Scouts, promoters, and connectors: The roles of ratings in nearest-neighbor collaborative filtering	recommender systems, recommendation, collaborative filtering
EComRec	E-Commerce Recommendation Applications	collaborative filtering, ecommerce, recommender
ContRec	A content-collaborative recommender that exploits WordNet-based user profiles for neighborhood formation	collaborative filtering, concept extraction, concept map, recommender
Group C		
GroupLens	Group Lens: An open architecture for collaborative filtering of netnews	collaborative filtering, recommender, recommendation
VirtCom	Recommending and evaluating choices in a virtual community of use	collaborative filtering, recommender
TagCF	Tag-aware recommender systems by fusion of collaborative filtering algorithms	tagging, recommender, collaborative filtering
TrustInRS	Trust in recommender systems	trust, recommender, collaborative filtering
RefWeb	Referral Web: Combining Social Networks and Collaborative Filtering	collaborative filtering, social network

Table 3.7: Users and groups in CiteULike.org.

to choose only one group. How does he decide which group fits more with his interests? The bibliography of a group contains several hundred of items, looking through them will take some time.

Let us assume that all tags are from the same taxonomy and there are no syntactical (e.g., tags *recommendation system*, *recommender systems*, *RS* are replaced with a single tag) and semantical (e.g., tags like *recommendation system*, *adaptive system* correspond to very same concepts in all bibliographies) inconsistencies in the names of papers and tags. Thus, we can represent a group or a user as a set of tags and a set of papers in their bibliography and calculate the degree of the fit between a user and a group as similarity between their sets of tags and papers. Moreover, we can see which papers are common for all three groups, creating for Michael a list of papers to read.

In this example we will consider only the sharing aspect of culture, as suggested in [154]. We argue that for defining a culture of a community in a Web 2.0 system it is enough to consider just the aspect of sharing for the two following reasons: 1) in this domain, measuring transmission is hard if not impossible. For instance, it is probably hard for anyone to recall how the ability of copy-paste fragments of texts using CTRL+C and CTRL+V is acquired — if they learned it from manuals, or from someone else; 2) since traits are transmitted by non-genetic means, they have been acquired during someone's life, so they were learned, or transmitted in another way, but not innate. For instance, it is hard to

imagine someone who knows how to copy-paste text since their birth. Consideration of only shared traits also allows for faster computation of the culture of a group.

3.6.2 Applying our approach

In our formalism, the users and groups are agents that are represented as a set of traits, which are papers and tags. For each agent, its culture is the set of traits as follows:

$Michael.papers=\{PolyLens, TrustInRS, GroupLens, RefWeb, TrustCF\}$
$Michael.tags=\{recommendation, collaborative\ filtering, trust, grouplens\}$
$GroupA.papers=\{EComRec, TechLens, GetToKnow, GroupLens, PolyLens\}$
$GroupA.tags=\{collaborative\ filtering, recommendation, academic\ reference, recommender, ecommerce\}$
$GroupB.papers=\{TechLens, Citations, Scouts, EComRec, ContRec\}$
$GroupB.tags=\{collaborative\ filtering, content\ based\ filtering, papers, citations, recommender\ systems, personalization, recommendation, ecommerce, recommender, concept\ extraction, concept\ map\}$
$GroupC.papers=\{GroupLens, VirtCom, TagCF, TrustInRS, RefWeb\}$
$GroupC.tags=\{collaborative\ filtering, recommender, recommendation, tagging, trust, social\ network\}$

Let us select one of the metrics from Section 3.5, say culture similarity, for determining how close are two cultures. Since the number of distinct papers in $Michael$'s and $GroupA$ bibliographies is eight, the number of common papers is two, the number of distinct tags is seven and the number of common tags is two, the similarity between $Michael$ and $GroupA$, $CS(Michael, GroupA)$ is equal to $0.5 \cdot \frac{2}{8} + 0.5 \cdot \frac{2}{7} = 0.268$. The similarity between $Michael$ and $GroupB$ is $0.5 \cdot \frac{0}{10} + 0.5 \cdot \frac{2}{13} = 0.077$, while the similarity between $Michael$ and $GroupC$ is $0.5 \cdot \frac{3}{7} + 0.5 \cdot \frac{3}{7} = 0.429$. From this simple exercise we can conclude that $Michael$'s research interests, as represented by his bibliography, are closer to $GroupC$. The program realizing such algorithm in real CiteULike.org settings, i.e. with hundreds of groups with thousands of papers, would solve the above-mentioned problem of choosing which community to join.

Let us further illustrate how our formalism can be applied to these data. Let us consider each group as an agent and see which traits are shared by the set of agents $\{GroupA, GroupB, GroupC\}$. Papers $EComRec, TechLens, GroupLens$ and tags $recommender, ecommerce, recommendation, collaborative\ filtering$ are weakly shared by the set and therefore are a culture of the set. Moreover, while there are no strongly shared papers, tags $collaborative\ filtering, recommender, recommendation$ are strongly shared and therefore are a strong culture of the set.

3.6.3 Discussion

In the case study we calculated the degree of culture similarity between Michael and different groups, and computed a culture of a set of CiteULike groups. These two problems fit our classification of problems involving culture, presented in Section 3.4. In particular, the calculation of similarity falls into *evaluate* class, problem class 25, while the discovery of culture classifies under *discover*, problem class 9.

Further extending this example, we might take into account not only artifacts such as papers or tags, but also behaviors of users, such as tagging some paper with a specific tag. For instance, using information about authors of the papers and citations, it is possible to consider behaviors such as self-citation and to see if there are communities whose members follow this practice more than an average author. Using information about the publication date and the date of posting the publication in someone's library it is possible to consider behaviors such as "tagging paper before its publication" and see which communities have the practice of dissemination of drafts of the papers.

3.7   Concluding remarks

In this chapter we have reviewed the notion of culture in the anthropology and social science, and, based on the literature, defined culture of a set of agents as a set of traits shared by the set of agents and transmitted. Moreover, using our definition it is possible to compute culture in a specific instant of time, or in the dynamics of several states. Based on our formalism, we have proposed a classification of problems that involve culture. This classification can be used to describe problems of discovering and evaluating culture in different domains. For the specified inputs, outputs, and constraints, the classification is complete. We also defined a set of metrics to evaluate culture in a fixed state and in dynamics. Through Sections 3.2, 3.3, and 3.5, we have used an example from anthropology to illustrate how our model of culture works. We have provided a Web 2.0 case study in Section 3.6 to show how our model can be used to compute and measure culture in modern systems for communities.

There are several issues we would like to underline.

First, the definition we provided here is operational in the sense that a computational model can be built on top of it and applied for computing culture in different domains, as illustrated by the running examples and the case study. This is possible because the formalism is based on set theory and this allows us to use the underlying mathematics to compute and measure culture.

Second, an important step in setting up our model is the definition of the level of granularity of different behaviors, which are used. For instance, it is important to decide if we should consider the behavior *eating* or several behaviors like *eating_with_sticks* or *eating_with_fork*. Another example could be — should we consider the behavior *open_window* or more specific behaviors like *turn_handle*, *pull_window*, etc. We leave the resolution of the issue of the granularity of behaviors to the domain expert.

Third, our model is limited by several assumptions on the nature of behaviors. First assumption is that there are no concurrent behaviors in the world, second assumption is that we consider only not innate behaviors as participating in transmission. However, these limitations only influence the notion of performing a behavior, while for computing culture we need to know about the fact that transmission took place, without going into details how it has been performed. Thus, this limitation somehow impacts only the way we can describe transmission, but hardly the model as a whole. Another assumption is that we do not consider traits attributed to the society as a whole, such as birth rate. We believe such traits can be integrated in our formalism later.

Finally, in Section 3.5 we have mentioned the idea of introducing domain-specific similarity function for traits. Such function would allow for considering the degree of similarity between different traits, e.g., specifying that trait *eating_with_sticks* is more similar to *eating_with_fork* than to *telling*. This approach can also be applied in domains with low cultural homogeneity between agents, i.e., the lack of shared traits, to introduce something like quasi-sharing (sharing similar, but not equal traits), and quasi-culture (a product of quasi-sharing). This is in line with Sperber's meaning of "shared" as that individuals belonging to a group have mental representations similar enough to be considered versions of one another.

In Section 3.4 we did not consider problem classes where the set of agents evolves over time, however, we define and address some of such problems with the framework presented in the next chapter.

Chapter 4

Implicit Culture Framework. Definition, Architecture, Implementation

In the previous chapter, we have proposed a definition of culture and shown how to use the proposed formalism to compute and measure culture of a set of agents. In Section 4.1 of this chapter, we use the proposed model to formulate a general problem of culture transfer. We then introduce a narrower problem of behavior transfer and propose the Implicit Culture Framework for solving this problem[1].

The Implicit Culture Framework is an agent-based framework that includes the following elements:

- a meta-model for defining the application domain;

- a general architecture of SICS for behavior transfer;

- a detailed architecture of SICS modules;

- algorithms helping the SICS modules to implement their functions;

- the IC-Service, a general-purpose, domain-independent service that implements the SICS architecture and the algorithms;

- a methodology that provides guidelines for applying the Implicit Culture Framework in practice.

The meta-model, presented in Section 4.2, is a refinement of the concepts of the formalism for representing culture, described in Chapter 3. The general and the detailed architecture of SICS for transferring behavior, are presented in Section 4.3 and Section 4.4, respectively. The implementation of the framework is presented in Section 4.5, while the methodology for its application are presented in Section 4.6.

---

[1]A paper derived from the content of this chapter has been published in the proceedings of ACM-SAC 2007 [16].

4.1   The problem of the transfer of culture

Let us consider two sets of agents, G and G' in a state s, and two corresponding maximal cultures: $T_G^{max}(s)$ and $T_{G'}^{max}(s)$. We do not impose any restrictions on the agents that are in G and in G', so G and G' can be the same, overlapping, or distinct. We use the maximal cultures of the sets, because they are uniquely defined. Without the loss of generality, let us assume that $T_G^0 = T_G^{max}(s) \setminus T_{G'}^{max}(s) \neq \emptyset^2$. Then, *the problem of culture transfer from G to G'* can be seen as the problem of transmitting T_G^0 to G' so that in some state s' after s, $T_{G'}^{max}(s')$ contains T_G^0. This problem is formulated as follows:

> *given two sets of agents G, G' and the maximal cultures $T_G^{max}(s)$ of the set G in the state s, $T_{G'}^{max}(s)$ of G' in the state s, find a state s' such that is_after(s', s), and $T_{G'}^{max}(s')$ contains $T_G^0 = T_G^{max}(s) \setminus T_{G'}^{max}(s)$.*

This problem somehow extends the example from problem class 21 in the classification of problems involving culture (Table 3.5), which is formulated as follows:

> given a set of agents G and a culture T_G of the set G, find a state s' such that T_G is a culture of G in the state s'.

With respect to problem class 21 we need to find a state s' such that T_G is a culture of another set of agents G'.

The problem of preserving a specific culture of the set is a particular case of the problem of culture transfer, where G is the initial set of agents in the state s, G' is the same set of agents in the state s', and T_G^0 is defined as above, or given as a subset of $T_G^{max}(s)$ that must be preserved. This problem is formulated as follows:

> *given a set of agents G in a state s, the same set in a state s', is_after(s', s), denoted as G', and a culture T_G of the set G in the state s, preserve T_G as a culture of G' in the state s'.*

This problem extends examples of problem classes 23 and 24 in the classification of problems involving culture (Table 3.5). Example for problem class 23 is formulated as follows:

> given a set of agents G, a state s, and a culture T_G of the set G in s, preserve T_G as a culture of G also in the next state s'.

Problem class 24 extends problem class 23 with specifying an evolution of culture:

> given a set of agents G, a sequence of states $\{s\}$, and an evolution of culture T_G of the set G in the subset of the sequence of states, $\{T_G\}$, preserve T_G as a culture of G also in the other states of the sequence.

With respect to problem classes 23 and 24, the problem of preserving a culture of the set assumes that the set of agents G changes since the state s and becomes G' in the state s'.

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*

---

[2] If $T_{G'}^{max}(s) \setminus T_G^{max}(s) \neq \emptyset$, then we can swap G and G', while if $T_G^{max}(s) = T_{G'}^{max}(s)$ then we can consider strong cultures of the sets in the hope that they differ, but it is important to have this non-empty difference of cultures if we speak about culture transfer, otherwise there is no problem of transfer.

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 T_G^0 from G to G' may result in transferring a bigger set of traits, $T_G'^0$ such than $T_G^0 \subseteq T_G'^0$. An example further supporting our argument can be find in [87, pp. 26,30], where the authors observed that by learning English, Arab students learned "something else", namely some implicit elements of Western culture.

Definition 14 (implicit culture relation) *A set of agents G' in a state s' is in 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',*

- *agents of G' do not perform explicit actions to acquire traits from T.*

By the last items 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'.

Now we would like to recollect the distinction between *action* and *behavior* we discussed in Section 3.3. In AI literature, an action is an atomic piece of activity, while behavior is perceived as something more complex, and can include several actions. Previously, we said that in a general case, it is hard or even impossible to represent relation between behavior and actions. However, in the following we try provide such representation for a particular case.

Let us consider a sequence of actions $\alpha_1, \alpha_2, \ldots, \alpha_k$. We can consider a behavior τ_1 that corresponds to this sequence. A behavior τ_2 corresponds to another sequence of actions $\alpha_{k+1} \ldots \alpha_n$. Let us assume that $\tau_1 \to \tau_2$ (τ_1 implies τ_2) for some set of agents G. This means that τ_1 is always followed by τ_2. Let us consider behavior representing this implication for each member of community: $\tau_G = \tau_1 \to \tau_2$ can be considered as composed of τ_1 and τ_2. *The Implicit Culture Framework, presented in the following sections, focuses on this narrower problem of transferring behavior in the form $\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_k \to \alpha_{k+1} \wedge \ldots \wedge \alpha_n$ from a set of agents G to a set of agents G'.*

If a trait τ transferred from G to G' in some state preceding s', we assume that it was transmitted, and therefore $transmitted(a, a', \tau, s')$ holds for some $a \in G$, $a' \in G'$. Since the trait is transmitted, for τ being in a culture of G' it is enough that it is shared by G'. We show that it is possible to achieve such sharing and transmission using our approach, and, therefore, to achieve the implicit culture relation between G' in s' and G in s for a set of traits T. We call the architecture that helps to establish this relation between two sets of agents a *System for Implicit Culture Support*.
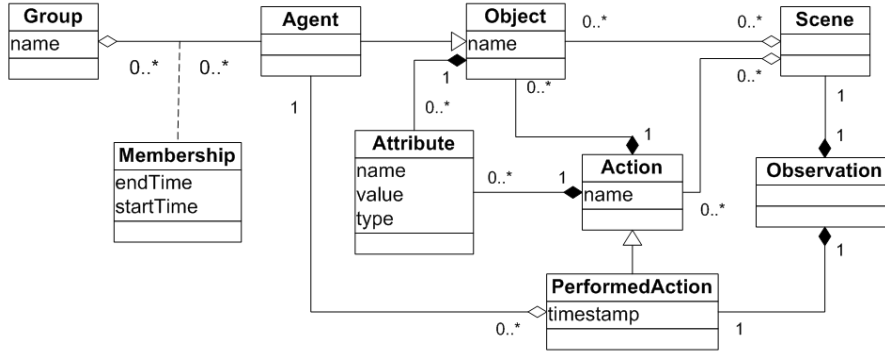
Figure 4.1: The meta-model of the Implicit Culture concepts.

4.2   Meta-model

The meta-model that illustrates relations between the core Implicit Culture concepts is shown in Figure 4.1. An environment is described in terms of *agents* which 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:

$$\texttt{action\_name}(\texttt{agent\_name}(ag\_attribute\_name_1 = ag\_attribute\_value_1);$$
$$\texttt{object\_name}_1(o\_attribute\_name_1 = o\_attribute\_value_1),\dots; \qquad (4.1)$$
$$attribute\_name_1 = attribute\_value_1,\dots; timestamp),$$

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. Full syntax of the language we use to represent actions is given in Appendix A.

In relation to the formalism we present in Chapter 3, the concept of *scene* we introduced here, is an abstraction for representing "similar" states. Here by similarity of the states we mean that the same or similar behavior can be performed in those states. Assuming that behavior depends on the context, which is the part of environment faced by agent performing behavior, we use scenes to represent the context. Behavior in such formulation is similar to the concept of situated action [138]. In such formulation, an observation puts a performed action in relation with the scene, i.e. the context in which the action has

been performed. Thus, in the following we can carry out analysis of which actions are performed in which scenes.

Let us consider a scene $c = (\alpha_1, \ldots, \alpha_k; o_1, \ldots, o_l)$ 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 15 (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.

4.2.1   Cultural theory

As we mentioned previously, the Implicit Culture Framework transfers behavior in the form $\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_k \rightarrow \alpha_{k+1} \wedge \ldots \wedge \alpha_n$. To represent the behavior that is subject to transfer from G to G' we introduce the notion of *cultural theory*.

The cultural theory, denoted as θ, is expressed by a set of rules of the form:

$$A_1 \wedge \cdots \wedge A_n \rightarrow C_1 \wedge \cdots \wedge C_m. \tag{4.2}$$

Here $A_1 \wedge \cdots \wedge A_n$ is the *antecedent* and $C_1 \wedge \cdots \wedge C_m$ is the *consequent*. Each element of the antecedent and of the consequent is either an action α represented as in Equation 4.1, 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 in Section 4.5.

Definition 16 (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.

4.3   General architecture of a System for Implicit Culture Support (SICS)

The general architecture of a SICS is shown in Figure 4.2 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*;
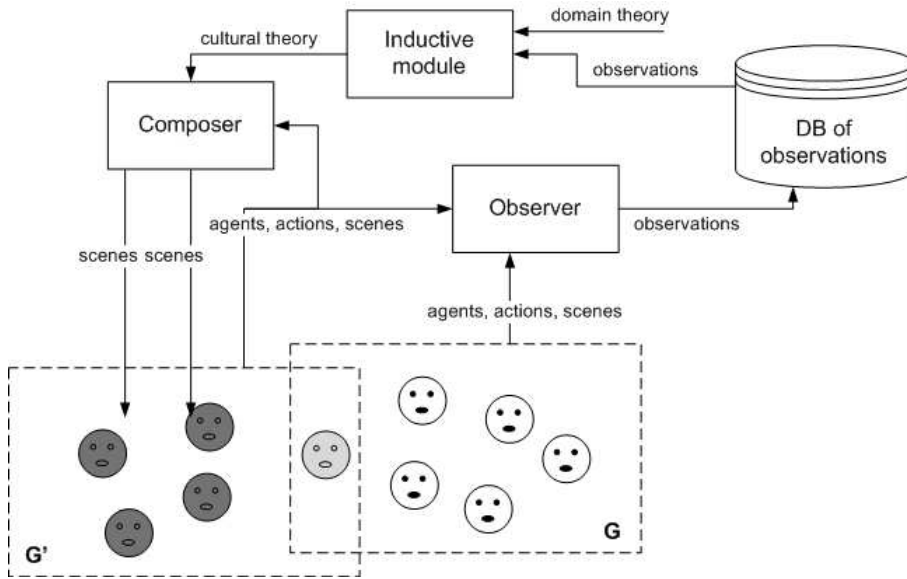
Figure 4.2: The architecture of a SICS. The SICS includes three components: the *observer*, which monitors agent activities and stores in the database (DB) the observations about the performed actions; the *inductive module*, which discovers cultural theory about agent behavior by analyzing observations; the *composer* module, which uses observations and the cultural theory to manipulate scenes so as to change behavior of another set of agents as required by the 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 traits T. 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 set of traits T to be transferred is not pre-defined, but must be discovered. The proposed SICS architecture of 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 behavior traits to be transferred from G to G'. The second part is learned by inductive module. In the example of newcomer in organizational settings by Wyatt-Haines [154], given in subsection 2.3, θ_0 corresponds to the knowledge about systems and processes the key people the newcomer gets explicitly when starts working (explicit culture, in terms of Section 2.2). The other part of the theory corresponds to that part of the culture which is left implicit, and must be learned by the newcomer alone (implicit culture).

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.

In the following section we present a detailed SICS architecture elaborating in more details the second step of achieving the implicit culture relation.

4.4 Detailed architecture of a SICS

In this section, we describe the composer module of the general SICS architecture in detail and present algorithms used in the composer.

Returning to the second step of achieving the implicit culture relation by SICS, 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 4.3:

- The *Cultural Actions 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, then 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 w.r.t. the goal of having a specific action performed, namely the *persuasiveness* of the scene. The scene producer problem is different from classical supervised or unsupervised classification problems and clustering.
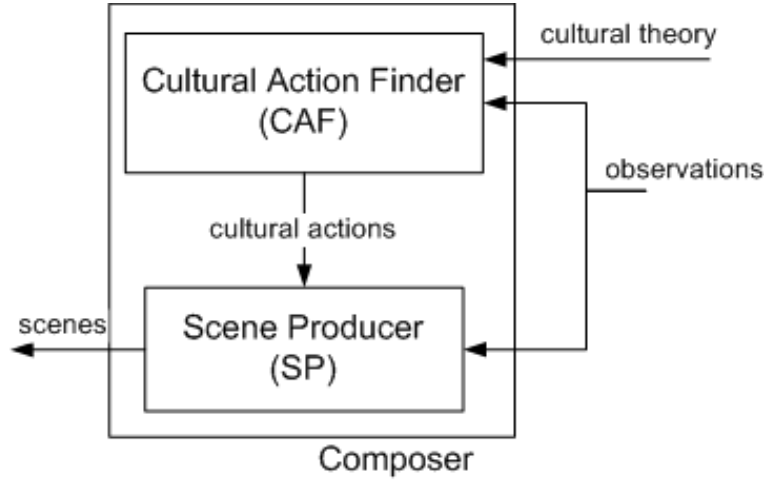
Figure 4.3: The internal architecture of the composer module. In the first step, the composer looks through observations to select an action that matches the antecedent of the rules of the cultural theory. In the second step, the *CAF* produces cultural actions corresponding to the rule fired in the first step. In the third step, the *SP* produces scenes in which the cultural actions are likely to be performed.

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

4.4.1   Cultural Actions 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 4.4 presents the algorithm of the CAF. For each rule `r` (ant→cons), the function $match(\rho,\alpha)$ checks whether the atom ρ of `ant`=$ant(\mathtt{r})$ matches the action α; then the function $find\text{-}set(\mathtt{ant},\mathtt{past\text{-}actions})$ finds a set of `past-actions` of past actions that match the set of atoms of `ant`; and finally, the function $join(\mathtt{past\text{-}actions},\mathtt{r})$ joins the variables of `r` with the actions in `past-actions`, i.e. it fills the corresponding variables in the rules with values from `past-actions`. The function $cons(\mathtt{r}')$ returns the consequent of the rule `r'`.

4.4.2   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:

1. 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;

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

```
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 4.4: The algorithm of the CAF submodule.

```
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 4.5: The algorithm for the first step in the SP.

3. 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 4.5 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). We describe examples of such a function in Section 4.5.

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:

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

(4.3)

where $C(a_1) \cap C(a)$ is the set of scenes in which both a_1 and a performed at least one action. $Ac_{a_1}(c)$ 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 4.3 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 $E(sim(\alpha_{a_1}, \alpha)|c)$, where α_{a_1} is the action performed by the agent a_1, α is the cultural action, and $c \in C$ is the scene in which α_{a_1} is situated. The following estimate is used:

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

(4.4)

that is 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 $E\left(sim(\alpha_{a_i}, \alpha)|c\right)$ with $a_i \in G_0'$ in Equation 4.4, to avoid recursion, estimated as follows:

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

(4.5)

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

The algorithms described above are fully implemented in Java using XML for expressing the cultural theory, as described in the next chapter. 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:

$$sim(a_1, a) = \gamma \cdot sim(a_1, a) +$$
$$+ \frac{1-\gamma}{|C(a_1) \cap C(a)|} \sum_{c \in C(a_1) \cap C(a)} \frac{1}{|Ac_{a_1}(c)||Ac_a(c)|} \sum_{\alpha_{a_1} \in Ac_{a_1}(c)} \sum_{\alpha_a \in Ac_a(c)} sim(\alpha_{a_1}, \alpha_a), \quad (4.6)$$

which is a modified Equation (4.3), $sim(a_1, a)$ 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 (4.3) is obtained from (4.6) taking $\gamma = 0$.

4.5 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.

We first present the architecture and the invocation scenarios of the IC-Service, then proceed with the description of modules of the IC-Service and discuss the implementation of the cultural theory.

4.5.1 The IC-Service architecture and invocation scenarios

The IC-Service architecture is based on the meta-model of the Implicit Culture concepts described in Section 4.2, and on the general and detailed SICS architectures described in Section 4.3 and Section 4.4. In the following, we describe in details the SICS implementation used in the IC-Service and justify why particular tools and architectures have been adopted.

The IC-Service is the remote part of the SICS which provides the access to the Implicit Culture Framework functionalities. The SICS architecture consists of three main layers (Figure 4.6):

- The *SICS Remote Client* provides a simple interface for the remote clients. It presents a wrapper that hides information exchange protocols.

- The *SICS Remote Module* defines protocols for information exchange with the client and converts the objects of the SICS Core into the format compatible with these protocols.

- The *SICS Core* provides the implementation of the Implicit Culture approach. This layer is responsible for storing observations, managing theory, and proposing scenes.

There are several ways the layers of the SICS architecture can be combined, allowing for the inclusion of the IC-Service in various applications ranging from small-size applications to complex distributed systems:

1. SICS can be included in the application as a library. In this case the *SICS Core* deals directly with the objects, actions, etc. of the application. This way should be chosen when the application is not necessarily distributed and can be tightly-coupled with the library.

2. To enable remote access, the SICS core can be invoked via the *SICS Remote Module* as a SOAP web service or an EJB component (using SOAP/RMI). This scenario should be adopted when the service is a part of a distributed system, but for some
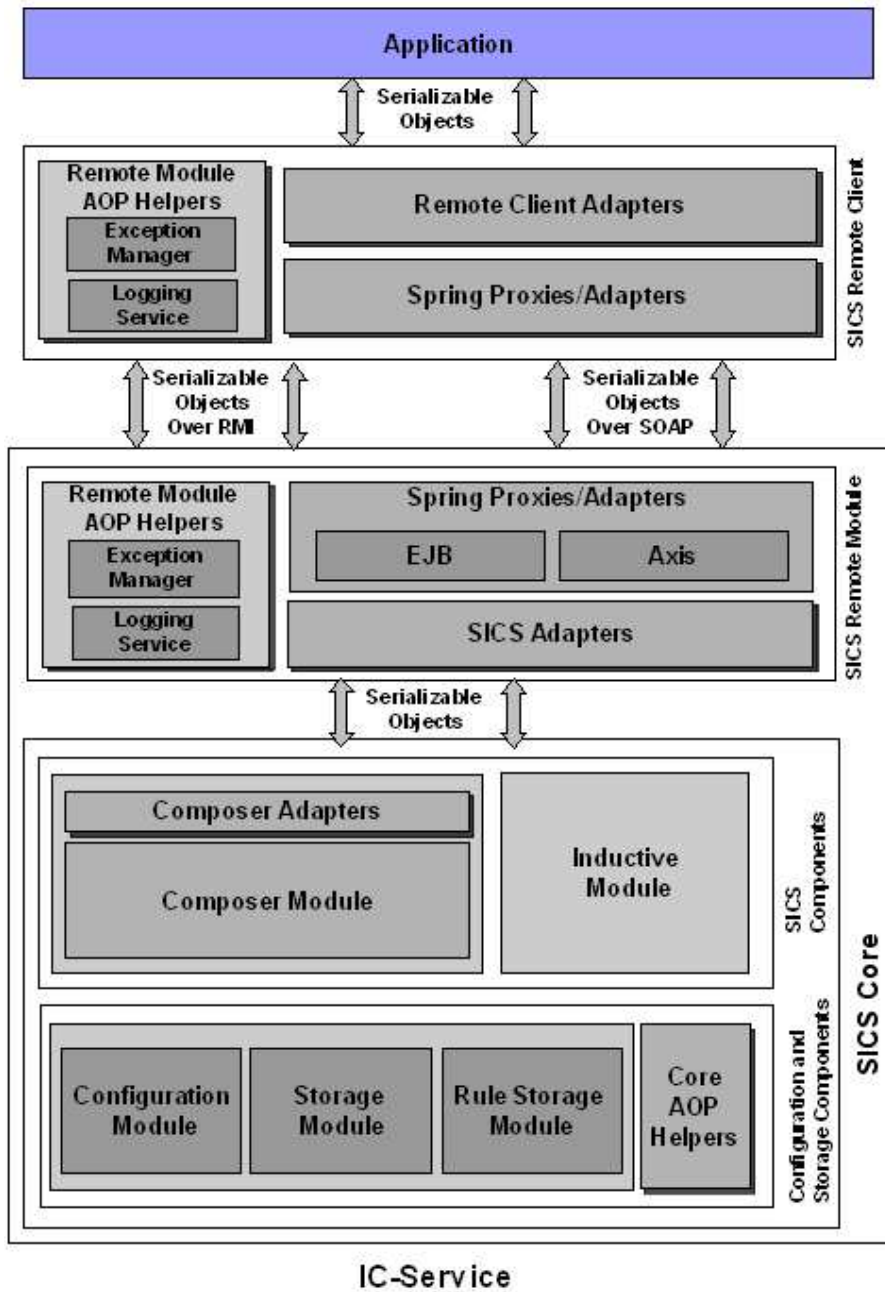
Figure 4.6: The detailed SICS architecture implemented as the IC-Service.

reasons there is no need or opportunity for using the *SICS Remote Client*. This may happen, for instance, when using the IC-Service in portable devices that have limited resources. However, in this case the application must take care of communicating with the service.

3. The easiest way to add recommendation service in an application is to access the IC-Service via the *SICS Remote Client* that hides the technical details of the communication mechanism from the application designer. This way should be adopted for using the IC-Service in complex applications in a fully decoupled way.

The described scenarios illustrate the possibility of including the IC-Service in various applications ranging from small-size applications to complex distributed systems. The IC-Service was developed using JAX-RPC (Java API for XML-based Remote Procedure Calls[3]), a programming model that enables invocation of web services across heterogeneous platforms. The SICS modules are built using the Spring framework[4], which allows assembling of loosely-coupled components in a complex system via XML configuration files. All modules apart from the Storage Module and the Rule Storage Module communicate through Java function calls and serializable objects. By avoiding Java collections, the easier interoperability with SOAP is enabled.

SOA has been chosen among the possible architectures because it supports principles of universal access and platform independence and allows the IC-Service to be transparently located inside or outside the enterprise. Support of EJB technology simplifies the use of the IC-Service in applications developed with Java technology. The Storage Module supports two possible storage facilities: XML files and the database storage. XML files provide a simple, easily deployable, and portable solution for applications where the observation history is not big and must not be accessed frequently. The database option should be chosen with more complex applications involving heavy data processing.

In the following we describe the modules of the architecture in detail.

The SICS Remote Client.

The main purpose of the SICS Remote Client is to provide a simple interface for applications that access the IC-Service remotely. It is composed of *Remote Client Adapters*, *Spring Proxies/Adapters*, and *Aspect-Oriented Programming tools (AOP Helpers)*. *Remote Client Adapters* are responsible for the asynchronous invocation of the SICS Remote Module. *Spring Proxies/Adapters* provide the connection with the SICS Remote Module via SOAP or RMI(Remote Method Invocation). *AOP Helpers* provide logging, validation and exception management. SOAP[5] is a lightweight XML-based protocol for exchanging information in a distributed environment.

The SICS Remote Module.

The goal of the SICS Remote Module is to define protocols for information exchange between the SICS Core and the client, and to provide the conversion of SICS Core objects

---

[3]http://java.sun.com/webservices/jaxrpc/
[4]http://www.springframework.org
[5]http://www.w3.org/TR/soap/

into the format specified by these protocols. The SICS Remote Module includes the *Spring Proxies/Adapters* for the remote invocation of the modules of the SICS Core using SOAP or RMI. The use of *Apache Axis* in addition to the Spring framework allows the SICS Core to be available as a SOAP web service. *EJB* part of the Remote Module allows for the use of the SICS Core modules as an EJB components in J2EE environment. *SICS Adapters* provide the connection between the SICS Remote Module and the SICS Core. Finally, *AOP Helpers* deal with logging, validation and exception management.

The SICS Core.

The SICS Core implements the detailed SICS architecture, providing the means for managing observations, the cultural theory, and recommendations. **Composer Adapters** are auxiliary modules, in particular, responsible for the asynchronous execution of the Composer services and cache management. The main functionality of the **Composer Module** (Figure 4.7(a)) 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 4.7(b)) incorporates the implementation of the *Apriori Algorithm* for the association rule mining [2] 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.

All parameters of a SICS instance are configured in the **Configuration Module** shown in Figure 4.7(c). Each instance of the SICS can have different configurations of the composer module (*Composer Module Constants*), the mechanism of processing the theory in the inductive module (*Inductive Module Constants*), and similarity algorithms (*Configuration of Similarity Functions*). The following two modules are responsible for the configuration of a SICS instance: the *XML Definition Loader*, which loads the configuration of the similarity algorithm from the corresponding XML file; and the *Simple Class Wrapper*, which loads the configuration of the similarity algorithm from the hierarchy of classes used by the Spring framework.

The details of the **Storage Module (Observer)** are shown in Figure 4.7(d). 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. *Storage Adapters* provide asynchronous execution of methods of the Storage Module and cache management. A powerful high performance query service for database storage is provided by the *Hibernate*[6] library. The Storage Module also includes a set of tools to work with an XML representation of the SICS information: *XQuery/XPath Utilities* are used to read data from an XML repository, *Java/XML Transformers* convert SICS objects into XML format and *JDom Utilities* deal with editing of XML files.

---

[6]http://www.hibernate.org/

(a) Composer Module

(b) Inductive Module

(c) Configuration Module

(d) Storage Module (Observer)

Figure 4.7: The architecture of the SICS modules implemented in the IC-Service.

Figure 4.8: The meta-model of the cultural theory.

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 Storage Module, however, the Rule Storage Module supports only XML storage facilities. **Core AOP Helpers** provide logging, validation and exception management.

4.5.2   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 Section 4.2. 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 4.8). A rule of the theory is defined in the form

$$\textbf{if } consequent \textbf{ then } antecedent,$$

where consequent and antecedent consist of one or several predicates. The intuition is that if consequent happened then antecedent will happen.

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

$$\textbf{if } press(\_a;stop\_button;;t) \textbf{ then } exit(\_a;next\_stop;;t+1)^7$$

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

$$\textbf{if } request(\_a;;request\text{-}params=...;t)$$
$$\textbf{then } rate\_high(\_a;recommendation;request\text{-}params=...;t+1)$$

---

[7]Please recollect that we are using syntax introduced in Section 4.2 and further explained in Appendixapp:LangSyntax.

```
<?xml version="1.0" encoding="UTF-8"?>
<rules xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="sics_rules.xsd">
  <rule identifier="ICPatterns">
    <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="pattern_name">*</attribute-rule>
              </attributes>
            </object-rule>
          </objects>
        </action-rule>
      </action-predicate>
    </consequents>
  </rule>
</rules>
```
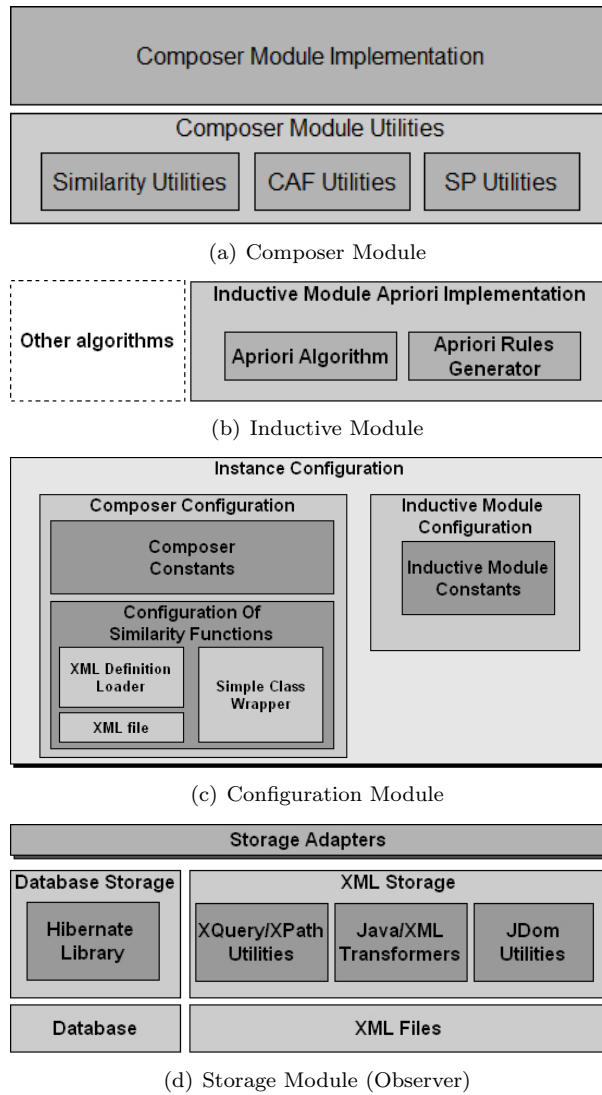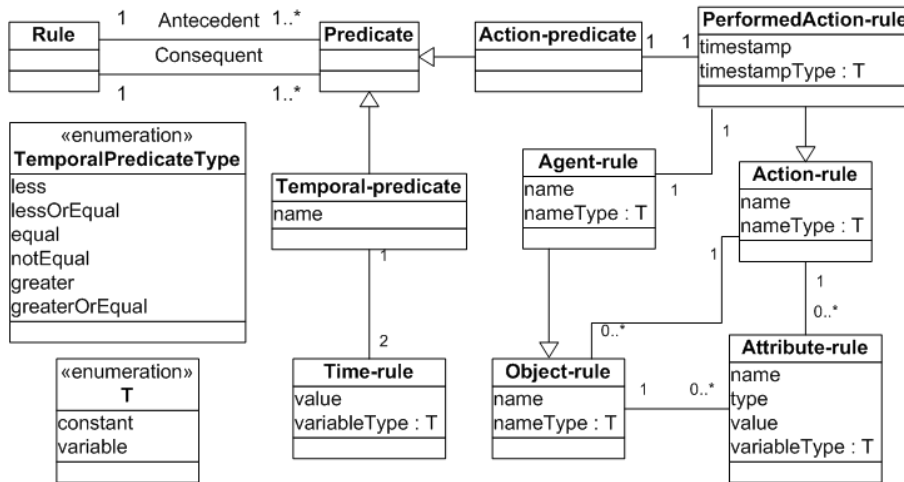
Figure 4.9: An example of the XML representation of the cultural theory in the IC-Service.

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*.

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 4.9. It can be represented in the language we use as:

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

and means that **if** someone is requesting [pattern] for the problem _x described by the attribute *keyword = _y*, **then** the returned pattern (specified with the attribute *pattern_name = **) is applied for the problem _x. This theory is used in the application of the Implicit Culture Framework to the software pattern selection, described in Section 5.2.

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 Section 4.4.

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 the thesis, but the algorithm is conceptually similar to the one described by Spanoudakis and Constantopoulos [134].

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 system described in Section 5.3, some attributes were compared using WordNet-based similarity metric, while in the IC-Patterns system described in Section 5.2 an ad-hoc algorithm for calculating similarity between user queries has been used.

4.5.3 Developing recommendation systems using the IC-Service. Lessons learned

Existing recommendation systems are usually tightly-coupled with the application domain, whereas recommendations services should be general, flexible, ubiquitous, and compositional. The IC-Service can be seen as a domain-independent solution for the development of general recommendation systems. The IC-Service has a flexible configuration mechanism that allows for its easy inclusion in different applications using web service technology. The recommendation algorithms rely on past user experience and they can be easily adapted to a specific domain in order to improve recommendation quality. Also, one of the problems is that recommendation systems developed so far mostly require explicit feedback (e.g. evaluation of items, relevance feedback). The use of the IC-Service in such applications allows for implicit feedback collection, which is now used only in some systems. We have developed two recommendation systems, described in Section 5.2 and

Section 5.3 of Chapter 5, using the IC-Service. In this section, we briefly summarize the experience we got from these applications.

Web service technology simplifies the development of recommendation systems and allows for the integration of the recommendation service to existing systems. However, there are several open questions regarding the design of services to be used as *long-lived* loosely-coupled components of distributed systems. What makes the IC-Service different from standard information services such as book-selling service is that it (i) is oriented on the use in various application domains, (ii) processes client data according to the rules defined for a particular application domain, (iii) supports storage of potentially huge amount of clients' data, (iv) analyzes the collected information in order to adapt the provided functionality to the needs of a particular client.

The principles underlying the design of such services are not well-established yet. Curbera et al. [39] describe *customization* of SOA components as one of the key characteristics. They argue that "a SOA programming model should enable building services and modules that programmers can customize without source code modification". Indeed, it is unlikely that a service can be reused by different applications without re-configuration. For its nature, the IC-Service has a direct dependence on the domain of the application and must be customizable. Therefore, configurability and extensibility without code modification were the main focus of the design process. To reach the necessary properties such as adequate level of granularity, flexible configuration mechanism, powerful storage and data management facilities, etc., we used state-of-the-art tools and solutions, namely, the combination of the original Implicit Culture theory with design patterns ("Adapter","Proxy", "Facade", "Abstract Factory", "Factory Method", etc.) [56], Aspect-Oriented Programming and auxiliary frameworks such as Spring and its principle of "designing to interfaces".

Multilevel organization of features and support of both XML and database storages are involved to satisfy the *portability* and *scalability* requirements. XML storage format imposes restrictions on the number of observations that can be stored. These restrictions can be overcome using database storage or deploying several instances of the IC-Service. To increase the performance, operations responsible for storing observations run in separate threads or JMSs (Java Message Services) under J2EE environment. Independent and configurable cache[8] is used at each functional level.

4.5.4 Implementation and integration details

The IC-Service has been developed with JAX-RPC, a programming model that enables invocation of web services across heterogeneous platforms. The SICS modules are built using the Spring framework, which allows for assembling of loosely-coupled components in a complex system via XML configuration files. All modules apart from the Storage Module and the Rule Storage Module communicate via Java function calls and serializable objects. Support of EJB technology simplifies the use of the IC-Service in Java applications.

The IC-Service can be added in an application in a fully *decoupled* way, and accessed from anywhere at any time. This guarantees *ubiquity*, allowing the system to process data from from different sources. For instance, ubiquity is very useful in the problem

---

[8]http://ehcache.sourceforge.net/

of providing cross-selling recommendations. Several communicating IC-Services can be seen as building blocks in the development of an efficient and robust decentralized system. At the same time, the IC-Service is a general-purpose and domain-independent application that provides means for storing, analyzing and reasoning about the observed behavior. It presents a higher *granularity* than specialized recommendation modules. Once deployed, the IC-Service can be used by several applications. Changes and extensions can be smoothly embedded in the working system by modifying XML-based domain description or the theory. This leads to minimizing efforts on development and reducing overheads on support of heterogeneous systems.

4.6   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 Chapter 5.

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 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 applications described in the following chapter, we explain these steps in more detail with examples (Sections 5.2, and 5.3).

4.7 Concluding remarks

Let us briefly summarize the main ideas of this chapter. The Implicit Culture Framework is an agent-based framework that includes a meta-model for defining the application domain, a general architecture of SICS for behavior transfer, a detailed architecture of SICS modules, and algorithms for their functioning. It also includes the IC-Service, a general-purpose, domain-independent service that implements the SICS architecture and the algorithms. Finally, it includes a methodology that provides guidelines for applying the Implicit Culture Framework in practice.

The current implementation of the framework focuses on a narrow problem of transferring specific traits that represent implication relationships between actions of agents. When the transfer of these traits takes place and the set G' starts behaving similarly to G, we call this the implicit culture relation. We argue, that even that technically the Implicit Culture Framework transfers only behavior, in practice, a side-effect of such transfer might be the transfer of knowledge and experience.

We have presented the meta-model of the concepts used in the framework: agents, objects, attributes, actions, scenes, performed actions, observations, etc. Our notion of object as something agents act upon is very similar to the notion of artifacts in multi-agent systems, mentioned in Section 2.1. It thus satisfies the need for describing the object of agent physical action, posed by Omicini et al. [107]:

> A notion of agent tool, or artifact, is then required, which could allow a theory of agent physical action to be developed at the same level of refinement as the theory of agent communicative actions.

The SICS architecture is a part of the Implicit Culture Framework that performs the specified transfer of the set of traits. The transfer requires several steps: the traits are first represented as a cultural theory, and then the SICS manipulates with the scenes, i.e. contexts of agent actions, faced by the agents of G'. The purpose of the manipulation is to make agents behave according to the cultural theory. While in Chapter 5 we apply the SICS only with pre-defined cultural theories, the general SICS architecture includes the inductive module for learning the theory about the set G. In a way, similarly to the implicit-explicit culture dichotomy or duality, discussed in Section 2.2, we now transfer explicitly represented part culture, while the inductive module enables the transfer of the implicit culture, which is not immediately obvious or told to newcomers, as in example by Wyatt-Haines (see Section 2.3). The SICS architecture aims at achieving the implicit culture relation between G' and G for the set of traits T, but since in the algorithms of CAF and SP we speak about similarities and probabilities, in general, the SICS can achieve the implicit culture relation only with a certain degree of probability.

We have described the IC-Service, an implementation of the Implicit Culture Framework. As we further show, it can be used as a recommendation service. To the best of our knowledge, only the IC-Service provides a domain-independent recommendation service that can be added into an application using several invocation scenarios (Java library, an Enterprise JavaBeans (EJB) component, a web service). Moreover, by using ad-hoc similarity configuration, the IC-Service supports both collaborative filtering and content-based recommendation methods.

Finally, we have provided a methodology for applying the Implicit Culture Framework in practice.

Chapter 5

Applications of the Implicit Culture Framework

This chapter describes several applications of the Implicit Culture Framework in the field of recommendation systems.[1] In Section 5.1 we describe Implicit, a recommendation system for web search. In Section 5.2 we present IC-Patterns, a system that helps developers to select software patterns suitable for their design problems. In Section 5.3 we describe an application to the problem of web service discovery.

For each application we first provide a brief introduction in the domain, and more background, when necessary. Second, we describe how we applied the Implicit Culture Framework and the IC-Service (where applicable). We then proceed with the description of the implemented system and review the related work, where it deserves a separate section.

5.1 Web search

Internet contains a lot of answers to our everyday questions and search engines are aimed at helping us to find the answers in a set of relevant links. However, results produced by search engines are mostly impersonalized and satisfy needs of "average" users. If interests of a user are specific, the most relevant link might not be among the top ten shown by conventional search engines. As stated by Gori and Witten [61]

> [...] the need to protect minorities can only be addressed within new paradigms; new, personalized views of the web that supplement today's horizontal search services. Different users may merit different answers to the same query [...].

In the literature this problem is addressed using Internet agents, recommendation systems and community-based search. *Internet agents* monitor user browsing behavior, learn preferences and build profiles of users to assist in their web browsing [36, 89]. Coalitions of agents are also used for answering queries of single or multiple users [35, 100] and

---

[1]Papers derived from Section 5.1 have been published in the proceedings of AAMAS 2005 [17] and in the proceedings of the MAIRRS workshop at IJCAI 2005 [20]. A paper derived from Section 5.2 has been published in the proceedings of ISC 2006 [18]. Papers derived from Section 5.3 have been published in the proceedings of BIS 2007 [83] and IEEE Journal of Software [19].

specific mechanisms such as auction protocols and reward techniques are applied to implement collaboration among agents [150, 149]. In order to personalize recommendations, *recommendation systems* analyze user queries, the content of the visited pages, or implicit and explicit indicators of satisfaction in order to extract knowledge about user needs and patterns of behavior. Recommendation systems are usually classified as *content-based* systems, which analyze the content of web pages [36, 131, 142], *collaborative filtering* systems [70, 76, 84, 90], which produce recommendations based on the similarity of users, and *hybrid* systems that combine the two approaches [9, 28, 101]. Although groups of users can have common interests or deal with similar problems, Internet agents and recommendation systems usually focus on isolated users. Differently, in the research on community-based web search (e.g., I-SPY [130], Beehive [75], and other systems [4, 51]) the focus is on the preferences of the community rather than those of a single user.

In the majority of solutions developed to date, explicit feedback from the user is required. This means that after receiving search results users must evaluate them, e.g. by rating, or ranking. This requires an additional effort from users and, therefore, explicit feedback is often discouraged [120]. Furthermore, sometimes users are inconsistent in the explicit ratings provided [82]. All these suggests that implicit indicators of user interests should be exploited. Moreover, the study by Fox et al. [55] has shown that implicit measures can be a suitable alternative to explicit feedback.

Summarizing all things above, we see the need for systems supporting web search in communities of like-minded users with specific interests. Moreover, such systems should use implicit feedback where possible and provide means for sharing search experience with the community members, i.e. the content found relevant by someone should be immediately available for others submitting similar queries. The goal of such systems should be to improve the quality of web search for the community.

In this section, we present a multi-agent recommendation system called Implicit, which is intended to support the web search of communities of people working together (e.g. a project team, PhD students of the same department, a community of practice). Such communities have specific common interests common interests related to their activities. Even though Web 2.0 provides a lot of tools for representing explicitly such communities (Facebook, LinkedIn, to name a few), these tools not necessarily provide support for web search. Our system is intended to be used in such communities for the purpose of sharing their search experience. The system can be used in order to increase quality of search in small communities, in terms of precision and recall, by supporting collaboration of the community members and sharing experience about using particular web links relevant to their specific interests.

The Implicit system aims at helping such communities to share their history of searches to recommend links relevant to their interests. Users submit their queries to the system and Implicit suggests specific links and people to contact. To produce recommendations relevant to community's specific interests, the system uses implicit feedback, namely, observed behavior of the members of the community. More specifically, it exploits previous observations about the behavior of other users after they submitted similar queries. Each user has a personal agent that interacts with the personal agents of other users to produce recommendations. The system implements a hybrid recommendation approach, provid-

ing users with the suggestions from and about the community members (collaborative recommendations) and with the results obtained from Google (content-based recommendations). The system allows for the exploitation of social interactions between community members, i.e. by their personal agents, in order to increase the quality of recommendations. Personal agents represent their users in the system, tracking their interests and browsing behavior with respect to using the links and contacts. Thus, Implicit also allows for shifting the burden of the collaboration task, namely, answering queries from other users, from the user to the personal agent of the user. The use of the Implicit Culture Framework helps new community members to behave similarly to the other members without the need of expressing explicitly the search behavior of the community.

5.1.1 Applying the Implicit Culture Framework

Let us map the web search domain to the terms of the Implicit Culture Framework. *Agents* are people searching the web. *Actions* are: *requesting* a link specifying a query, *accepting* or *rejecting* the proposed link. *Link* and *query* are objects. The object *query* has a *keyword* attribute. A *cultural theory* describing general behavior of the community in our system (the domain theory) is

$$\textbf{if } \text{search}(\_a; \_q(keyword = \_k); ; *) \textbf{ then } \text{accept}(\_a; *, \_q(keyword = \_k); ; *), \quad (5.1)$$

where $\_a$ is an agent, $\_q$ is a query, $\_k$ is a keyword and the wildcard next to $\_q$ represents a link. This domain theory (in this case, just a rule) is specified a priori and it says that if an agent $\_a$ searches with a query $\_q$ (keyword $\_k$) then the system should recommend some link that is likely to be accepted. In the Implicit system, the part of the cultural theory learned by the inductive module of SICS is the set of links accepted by the community for certain queries corresponding to their shared interests. Such theory represents the knowledge about user behavior that is learned by the system from user interactions with the system. An example of a such *cultural theory* describing actions of the community could be

$$\begin{aligned} &\textbf{if } \text{search}(\_a, \_q(keyword = \text{`apartments'}); ; *) \\ &\textbf{then } \text{accept}(\_a, \texttt{www.phosphoro.com}, \_q(keyword = \text{`apartments'}); ; *). \end{aligned} \quad (5.2)$$

This theory expresses that for all agents of the group if they search for apartments, they tend to accept the link `www.phosphoro.com`. This link is relevant to the specific interests of the group, in this case it is assumed that people search for the apartments in Trento, Italy and they would like to consider private offers, not those from an agency. This link is of extreme importance for people who have just arrived and search for an apartment in Trento, but they do not know about this website. For instance, in the time of writing this link did not appear among the first ten results provided by Google for the query 'apartments'. If personal agents of the newcomers are able to provide them with this link and they access the desired information, then it is possible to say that the new members behave in accordance with the community culture and that the implicit culture relation is established.

In the Implicit system each agent tries to establish the implicit culture relation within the group of agents on the platform. In order to do this, each agent rely on a SICS.

The observer module in the system monitors the actions users perform while interacting with the system. For instance, a query is treated as an information request. It is interpreted by the personal agent both as the request of a relevant resource link and as the request of the ID of an agent which can provide relevant recommendation. Therefore, two observations appear in the database of observations as the result of the query: `request(user,query,resource-link)` and `request(user,query,agent ID)`. If the user clicks on the recommended link, the link is considered to be accepted and the observation *accept(user, query, resource-link)* is stored in the database. If the *resource-link* has been suggested by an agent, which could be the user's personal agent or the personal agent of another user, one more observation is stored: *accept(user,query,agent ID)*. When the user starts another search or exits the system, all the recommendations which were proposed to the user but have not been accepted, are treated as rejected. For each rejected link two observations appear: *reject(user,query,resource-link)*, *reject(user,query,agent ID)*. It is important to notice that by storing IDs of the agents that provided the accepted or rejected recommendations the system can discover patterns of behavior related to accepting results obtained from a certain agent, thus maintaining implicit trust relationships.

The inductive module applies data mining techniques in order to extract interesting patterns from the user behavior. There are several approaches that can be exploited. Clustering can be applied in order to get knowledge about the correlations in the observations. For instance, agents can be clustered by interests and past actions of their users. Alternatively, we can apply association rules techniques, like Apriori [2] for learning association rules between the actions.

In the current version of the system the SICS implements the Apriori algorithm. This algorithm has been described by Agrawal and Ramakrishnan [2] and it deals with the problem of association rules mining. In our settings, this problem can be briefly formulated in the following way: given a database of queries and links, it is necessary to find which links are accepted for which queries. Without going into the details of the algorithm, we can say that mined rules have the form *query → link* and are characterized by confidence and support. The *confidence* of a rule denotes the fraction of cases when the *link* from the rule was accepted for the *keyword* from the rule. The *support* denotes the fraction of the actions in the database which contain this rule. Similarly, the problem for discovering which agents are accepted for which keywords can be formulated and addressed. Such problem is related to the problem of finding experts in a specific area of interests.

The SICS architecture allows the Implicit system to find relevant links and to discover IDs of relevant agents with the same mechanism. The SICS calculates the similarity between the community members in order to produce suggestions. Therefore, it personalizes web search to a certain extent.

5.1.2 The Implicit system

In this section, we describe the architecture of the system and the user interface. The details concerning the internal agent architecture and communication mechanism are given in Section 5.1.3.
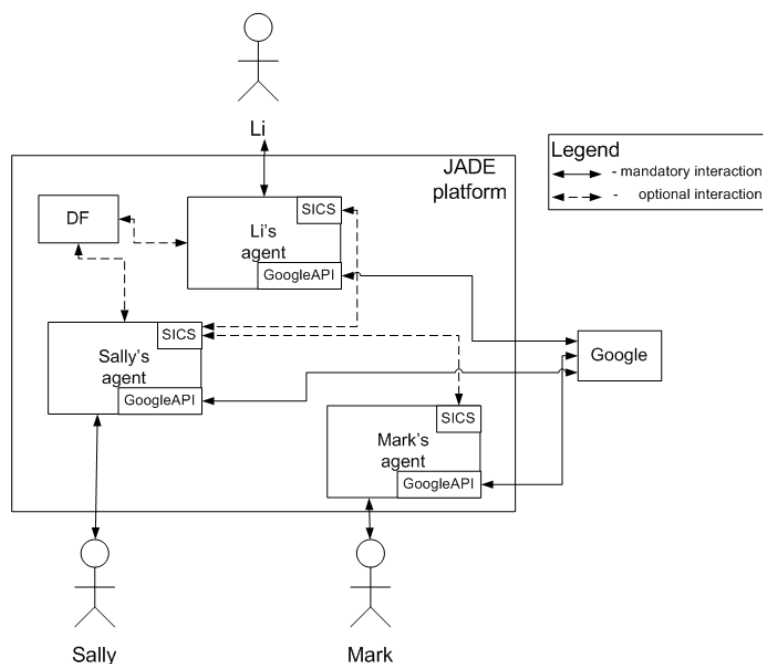
Figure 5.1: The architecture of the system. *Personal agents* process queries from *users* and interact with each other to share experience of using particular links by their users; the agents produce recommendations by using the *SICS* module; they also use *GoogleAPI* to query the Google search engine. The *Directory Facilitator (DF)* agent provides a list of personal agents.

The system architecture

Implicit is a multi-agent recommendation system that aims at improving web search of its users. The system has been implemented using JADE (Java Agent DEvelopment framework) [13]. JADE adopts a task-based model of the agent and it is a one of the most powerful tools for the development of FIPA[2]-compliant multi-agent systems.

Figure 5.1 depicts the architecture of the system. It consists of the client part and the server part. A user at the client side accesses an html/php user interface via browser. In the system, there is exactly one personal agent for each user. All personal agents are running on the JADE platform on the server side. The queries submitted by the user are received by the Java servlets which forward the queries to the user's personal agent. The personal agent uses its capabilities, described in more detail in the next section, to communicate with external information sources, e.g. Google, and to produce recommendations using its own resources and interacting with other personal agents. The obtained results appear in the user interface. When producing recommendations, agents aim at finding web pages that members of the community consider relevant to their searches. For this purpose, the agents adopt the Implicit Culture approach, searching for the links that satisfy specific behavioral patterns of the group.

Let us describe how users interact with the system. A user logs into the system, enters a query and receives the results from Google complemented with recommendations pro-

---

[2]`http://www.fipa.org/`. Foundation for Intelligent Physical Agents (FIPA).
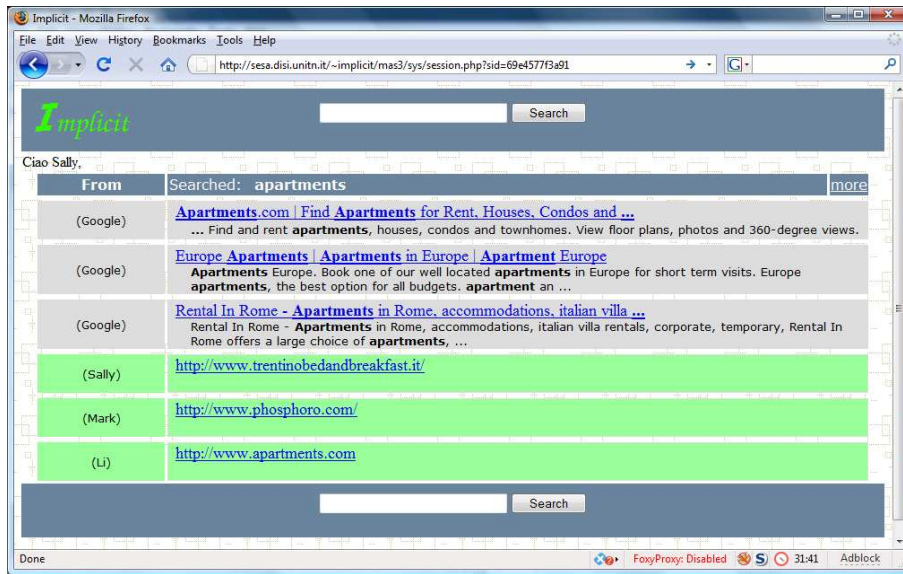
Figure 5.2: Results produced by the system for the entered query. The results from the Google search engine are displayed in the top part of the window. The links recommended by the personal agents of the other users are shown in the bottom part.

duced by the user's personal agent in collaboration with other personal agents. Figure 5.2 shows the browser window with the list of results. In the top part of the window there are the first ten links obtained from the Google search engine, while in the bottom part there are several links received as recommendations from the personal agents of the community members. The name of the link provider ("Google" or the name of the community member) appears in the box preceding the link. Whenever user clicks on one of the results, the information about this action is forwarded to the personal agent of the user as a *feedback* indicating relevance of the link to the search. After the user exits the system or starts another search, the non-clicked links are marked as rejected.

In the following, we will explain interactions between a user and a personal agent using a running example.

Example. Let us consider a user Sally who looks for a website that provides a collection of announcements about apartments available for rent. She logs in the Implicit system and types a query "apartments". The query is processed by the personal agent of Sally. Fist, the personal agent obtains the results from Google, and the ten results from Google are shown in the user browser. Figure 5.2 shows the following Google results: `www.apartments.com`, `www.only-apartments.com`, `www.rentalinrome.com`. Second, the personal agent uses the SICS module to process the query in several steps: searching for links during the internal search and searching for agents to contact during the external search. Searched links and agent IDs should be related to the entered query "apartments". If the agent does not find any agent IDs using the SICS, it contacts the Directory Facilitator (DF) agent (explained in more detail in the next section). Once the personal agent contacted all agents found during the external search or by contacting the DF, it displays the obtained links in the user browser. In this

example the links `www.trentinobedandbreakfast.it`, `www.phosphoro.com` and `www.apartments.com` from Sally and her colleagues Mark and Li are displayed. The personal agent stops the search at this point and becomes idle, waiting for the feedback or a new query from Sally and eventually responding to the queries of other personal agents. Let us suppose that Sally clicks on `www.phosphoro.com`. Her personal agent receives the feedback message about accepting this link. Since the link was suggested by Mark's personal agent, the feedback will be also treated as accepting Mark's personal agent. When Sally exits the system or starts another search, the not followed link, `www.apartments.com`, and the corresponding agent are marked as rejected.

In the current implementation, each agent uses the Google SOAP Search API to contact Google, but in principle, it is possible to contact any search engine that provides similar API. Implicit also allows for having some special agents on the platform, e.g. wrappers. Wrapper agents can be used for transferring the queries to other search engines like Yahoo! or Vivisimo[3].

Personal search history

Implicit also allows for the quick access to the history of previous user searches. The history is maintained by the personal agent in an XML file that contains links accepted in the past and the corresponding keywords. The agent accesses the history after querying Google and shows the results on the user interface. For instance, in Figure 5.2, the link from Sally's personal agent comes from Sally's history of previous searches. Another example of the knowledge available locally is a personal bookmark collection in someone's browser. User personal collection of bookmarks on Delicious could be an example of user-specific knowledge, which is not available locally, i.e. stored on the Internet.

Example. In our example, Sally history of previous searches contains the links `http://www.trentinobedandbreakfast.it/` and `http://www.suedtirolprivat.com/` for the keyword `bed&breakfast`.

Motivation for using agents in the system

The use of agents in the system is motivated by the following: i) agents assist their users in web search activities, i.e. agents personalize user searches, autonomously interact with other personal agents of the community, and facilitates maintenance of the past search history; (ii) agents provide an interface to kinds of search, i.e. Google, SICS, without the need of heavy client part of the system; (iii) agents recommend other agents on the platform thus establishing implicit trust relationships in the system; (iv) even if a user is not accessing the system for some time, the personal agent stays there, answers queries from other agents and improves its expertise; (v) agents facilitate sharing of information that is usually shared only by word-of-mouth communications; (vi) finally, in the simulations we conducted to validate the system (see Chapter 6), each agent contained a model of the user in order to simulate users of the system.

---

[3]A wrapper agent for the Vivisimo search engine has been developed as a student project at DISI, UNITN.
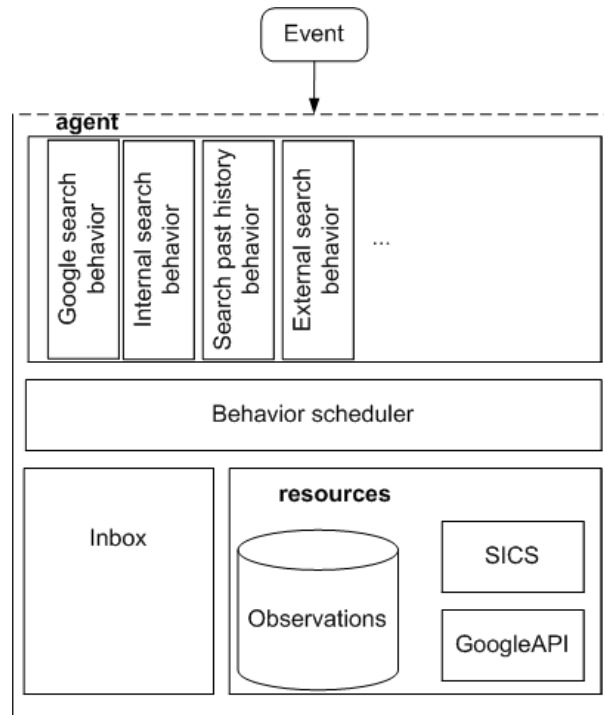
Figure 5.3: An internal architecture of the personal agent. A *behavior* of an agent is a task or reactions to an internal or external event. The execution of the behaviors and switching between them is performed by a *behavior scheduler*. An *inbox* contains ACL messages received by the agent. Agent's *resources* include observations, SICS and Google API.

5.1.3 Agent architecture and communication mechanism

This section provides more details about the technical description of personal agents, their interactions and the recommendation mechanism. We start from the description of the architecture of a personal agent, then we proceed with the description of the recommendation mechanism and protocols for the user-agent interactions. Finally, we present protocols for the communication and message exchange between agents in the system.

The architecture of a personal agent

In the following we define basic terms used in JADE and describe the internal architecture of an agent in our system. Figure 5.3 presents the internal architecture of a personal agent in the Implicit system and illustrates the definitions.

A *personal agent* is a software agent running on the server side assisting its user in their searches, receiving queries and producing recommendations in response.

A *behavior* is a procedure that implements tasks, or intentions, of an agent [13]. The agent is able to execute each task in response to different internal (for instance, calculations finished) and external (for instance, message received) events. Behaviors are logical activity units. They can be composed in various ways to achieve complex execution patterns and can be concurrently executed.

A *behavior scheduler* is an internal agent component that automatically manages the scheduling of behaviors and determines which behavior to run at the moment and what action to perform as a consequence.

An *inbox* is a queue of messages received from the user and from other agents. JADE agents use an Agent Communication Language (ACL) for exchanging messages.

To produce recommendations the agent uses its *resources* that include the information available to the agent, e.g., observations about user actions, and specific functionalities such as getting recommendations from the SICS or getting links from Google using Google API.

The search process

Let us describe behaviors and other parts of the agent architecture that participate in the search process in detail. As described in Section 5.1.2, a query received from the user interface triggers a set of steps executed by the personal agent. The process of producing recommendations that the user finally sees in the browser window consists of several parts, implemented as behaviors. When the agent receives the query message from the interface, it starts three search behaviors that run in the following order: first the *Google search behavior*, then the *Internal search behavior* that includes *Search past history behavior*, and, finally, the *External search behavior*. For brevity, we refer to the sequence of these three behaviors as "the search". The results obtained during all three steps of the search are shown to the user.

The sequence diagram in Figure 5.4 illustrates the details of the interactions between the user and the personal agent during the search. During the Google search behavior the agent forwards the query to Google search engine using Google SOAP Search API. After receiving the response, the agent shows the obtained links to the user and starts the Internal search behavior. In the internal search the goal of the SICS module is to recommend web links using the information about the past user actions about searches and link acceptance. In case the SICS does not produce any recommendation in this step, the past search history is used to recommend links accepted by the user for similar queries in the past. All the generated links are stored in the memory and the External search behavior is started. This behavior also uses the SICS, but the goal of the SICS in this case is to find relevant links using external resources, i.e. to propose the IDs of agents to contact. The techniques used within the SICS to recommend links and agents are the same. If there are no suggestions about agent IDs, the agent contacts the DF. According to the FIPA standards, the DF is a mandatory agent that provides yellow pages service on the agent platform. In our system, the DF simply provides the agent with the IDs of other personal agents on the platform. Thus, the use of the SICS module helps to reduce the number of interactions between the agents. Having filled the list of agents to contact, the personal agent starts interaction by sending a query to each agent in the list. When all the agents are contacted, the External search behavior queries new agents that were suggested during the search and so on. When all queries have been answered by the suggested agents, the system adds the obtained links to the list and shows all the links from the list to the user.

When agents query each other, the agent-responder does not contact Google, because
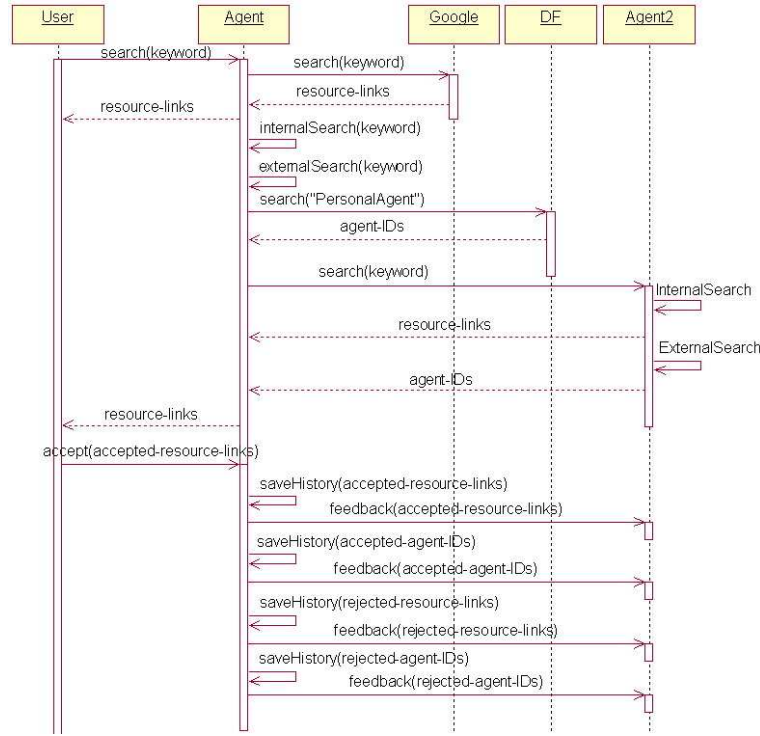
Figure 5.4: The sequence diagram of interactions between the user and the personal agent during the search.

the agent-questioner has this capability too. The agent-responder executes the Internal search behavior to produce links that the user of the agent-questioner will probably accept. The agent-responder also starts the External search behavior to recommend to the agent-questioner other agents to contact.

An example of recommendation

In this section, we provide more details on how recommendations are created. For explanations we will use the running example.

Example. Let us explain what happens when the personal agent of Sally receives the query "apartments" and starts the search. The following observation, which includes the type of the requested recommendation, i.e. link or agent, the name of the requester, and the query, is produced by the observer module of the SICS and appears in the database of observation: `request(Sally,apartments)`. This observation is then sent to the composer module that processes it in several steps. In the first step, the CAF builds the matrix of observations (Table 5.1) and matches the request action with the rule of the theory shown in Equation 5.1. The action matches the rule, so the right part of the rule, $accept(x, l, k)$, is taken as a cultural action. After substituting the value of the variables x and k with those from the request action, the cultural action $\alpha = accept(Sally, l, apartments)$ goes to the pool. The SP takes the action α from the pool and calculates which agents performed actions most similar to α. For this calculation, the SP uses the matrix of observations.

agent/link	www.hotel.it	www.phosphoro.com	www.only-apartments.com
Li			accept(apartments)
Mark	accept(hotels), reject(cars)	accept(apartments)	reject(apartments)
Sally	accept(hotels)		

Table 5.1: A matrix of observations. Rows contain users while columns contain links. Action performed by a user on a link are put in the cell on the intersection of the corresponding row and column.

The rows of the matrix contain agent names, and the columns contain links, while the entries contain actions that involve the corresponding agent-link pair, e.g. accept or reject of the link by the agent for a keyword. Since in the matrix of observations in our example (Table 5.1), Mark's actions are the most similar to Sally's actions, the link `www.phosphoro.com` is recommended and therefore put in the list of results.

Together with asking the SICS about relevant links, the personal agent of Sally submits another query to the SICS module, requesting agent IDs for the keyword "apartments", and the observation `request(agent,Sally,apartments)` is stored in the database of observations. Let us suppose that the SICS returns the ID "Li" as the result. The personal agent of Sally contacts the personal agent of Li and gets the link `www.apartments.com` as a recommendation. This link is put in the list of results and then the results, i.e. `www.phosphoro.com` and `www.apartments.com` are displayed in the user browser. The personal agent stops the search at this point and becomes idle, waiting for the feedback or a new query from Sally and eventually responding to the queries of other personal agents. Let us suppose that the users clicks on `www.phosphoro.com`. The personal agent of the user receives the feedback message that is converted to the action `accept(Sally, www.phosphoro.com, apartments)`. When Sally exits the system or starts another search, the feedback about the not followed link is received about the personal agent and converted to the action `reject(Sally, www.apartments.com, apartments)`.

Interactions between system components

Table 5.2 lists the details of interaction between the system components. The table lists the participants of an interaction in the columns Component1 and Component2, the corresponding actions, parameters and the desired result of the interaction (column Target). The last column shows which tools and communication protocols are used for the interaction. Here we briefly describe how components interact and in the following two subsections we provide more details on the user-agent and agent-agent interactions.

The interaction between the user and the personal agent is mediated by user browser, Java servlets and sockets. Therefore, actions and protocols are listed for the interaction between the servlet and the agent. The details of these protocols are described in the following subsection.

Personal agents communicate with the Google search engine using the Google SOAP Search API that allows one to query Google and get the first ten results from a Java program. The agent-Google interaction includes a request action, which starts a Google search with keywords passed as the parameter of the action, and an inform action, which

Component1	Component2	Action	Target	Parameters	protocol or tools of communication
servlet	agent	request	links	query	sockets,
					User-Agent Query Protocol
agent	Google	request	links	query	GoogleAPI
Google	agent	inform		links	GoogleAPI
agent	servlet	inform		links	User-Agent Query Protocol,
					servlets, sockets, browser
agent	SICS	request	links	request-type, query	Java class method call
agent	SICS	request	agent-IDs	request-type, query	Java class method call
agent	DF	request	agent-IDs	——	Java class method call
DF	agent	inform	——	agent-IDs	Java class method call
agent	agent2	request	links	query	Agent Query Protocol
agent	agent2	request	agent-IDs	query	Agent Query Protocol
agent2	agent	inform		links	Agent Query Protocol
agent2	agent	inform		agent-IDs	Agent Query Protocol
agent	servlet	inform		links	User-Agent Query Protocol
servlet	agent	inform		accepted-links	sockets
					User-Agent Feedback Protocol
servlet	agent	inform		link	User-Agent InsertLink Protocol
servlet	agent	request	links	query	User-Agent MoreResults Protocol
agent	SICS	inform		accepted-links, query	Java class method call
agent	SICS	inform		accepted-agent-IDs, query	Java class method call
agent	SICS	inform		rejected-links, query	Java class method call
agent	SICS	inform		rejected-agent-IDs, query	Java class method call
agent	agent2	inform		accepted-links, query	Agent Feedback Protocol
agent	agent2	inform		accepted-agent-IDs, query	Agent Feedback Protocol
agent	agent2	inform		rejected-links, query	Agent Feedback Protocol
agent	agent2	inform		rejected-agent-IDs, query	Agent Feedback Protocol

Table 5.2: The scheme of interactions between the system actors during a search session. *Component1* communicates with *Component2* performing the communication act *Action*; Component1 would like to obtain *Target* as a result of the communication; Component1 provides *Parameters* to Component2; the last column represents the protocol or tool used for the communication.

delivers the results of the search to the agent.

Each agent invokes the SICS using an appropriate Java class method. This interaction is performed using a request action with the type of the request (link or agent-ID) and the query being parameters of the request. When the agent communicates the user feedback to the SICS, an inform action with the accepted or rejected link and the corresponding query is used.

Interaction between personal agents and the DF agent are implemented using Java class methods provided by JADE. A request action is used to obtain the IDs of the personal agents on the platform, while the results are communicated by the DF using an inform action.

Personal agents interact with each other using several protocols described in detail in the following subsection. These interactions are mediated by the JADE platform that provides facilities for the communication between agents.

User-agent interactions. The interaction between the user interface and the multi-agent platform is performed in the following way: Java servlets on the client side communicate with the personal agent on the server side using sockets. The protocols used for such communications are shown in Figure 5.5, while the structure of messages and their content are explained below.

Figure 5.6 shows the structure of the messages used in the communication between the user interface and the personal agent. The structure is expressed in the Backus-Naur form. The `message_type` field illustrates the purpose of the message and can be
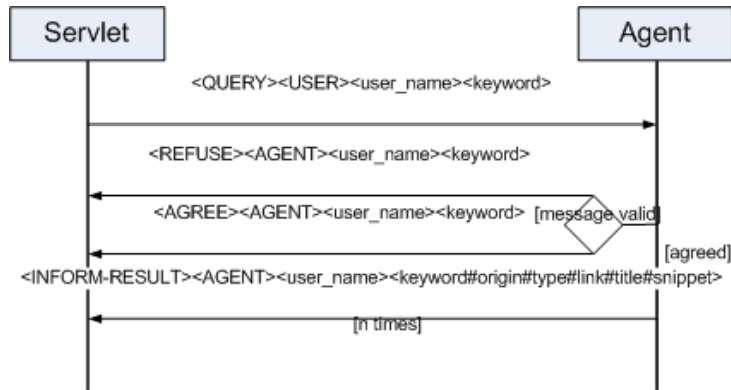
(a) User-Agent Query Protocol
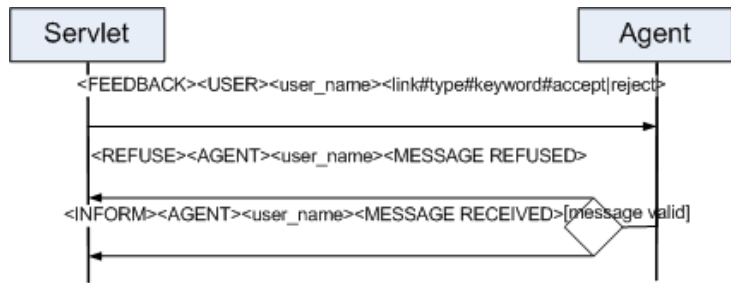
(b) User-Agent Feedback Protocol
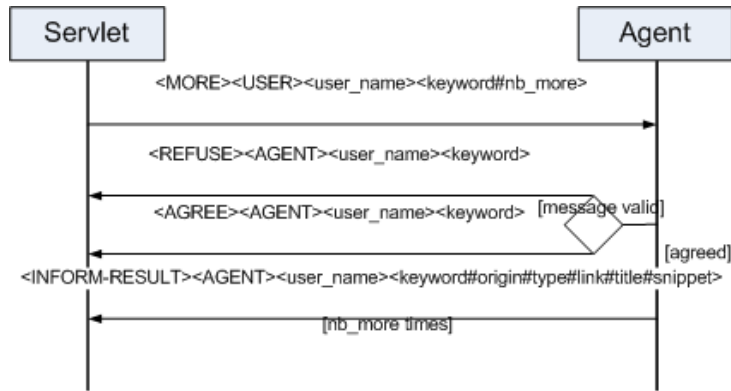
(c) User-Agent MoreResults Protocol
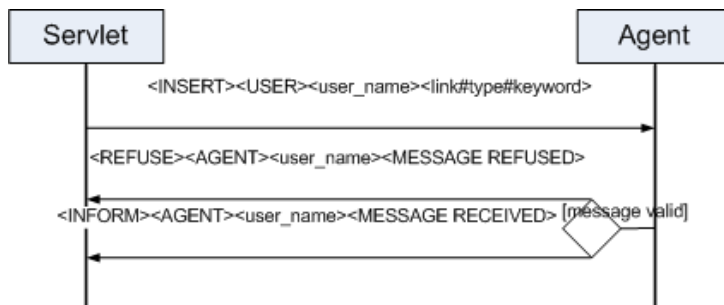
(d) User-Agent InsertLink Protocol

Figure 5.5: The protocols used for the communication between the user interface and the personal agent.

```
<message> ::= <message_type><type_of_communication><user_name><content>
<message_type> ::= QUERY|ACCEPT|MORE|INSERT|AGREE|INFORM|INFORM-RESULT|REFUSE
<type_of_communication> ::= USER|AGENT
```

Figure 5.6: The structure of the messages used in the user-agent communication.

one of the following types, explained in more detail below: *query*, *accept*, *more*, *insert*, *agree*, *inform*, *inform-result*, and *refuse*. The `type_of_communication` field indicates if the communication takes place between two agents (value $AGENT$) or between a user interface and an agent (value $USER$). The `user_name` field contains the name of the user that accesses the system, while the `content` field contains different information depending on the type of the message.

The User-Agent Query Protocol depicted in Figure 5.5(a) is used for submitting query from the user interface to the personal agent. In a way it is similar to the FIPA Query Interaction Protocol [4]. The protocol starts with a *query* message that contains the entered query. As a response, the agent can send either an *agree* message, if the message is valid, or a *refuse* message if for some reasons the agent cannot process the query. Sending the refuse message ends the interaction. The agree message means that the agent accepts the query and will send the query results to the user interface later. Each *inform-result* message contains the query the result corresponds to, the link that is relevant to the query, and some internal information such as the origin of the link, a short description, etc.

After viewing the results in the browser window, the user can request more results related to the same query. In this case, the User-Agent MoreResults Protocol is used for the communication between the servlet and the agent (Figure 5.5(c)). It starts with a *more* message that shows that the user requested more recommendations on the currently performed search. The parameter "nb_more" is the number of additionally requested recommendations. The rest of the protocol is the same as in the case of the User-Agent Query Protocol.

When the user chooses some links, i.e. clicks on it, the User-Agent Feedback Protocol (Figure 5.5(b)) is used in order to pass the feedback about an accepted or rejected result to the agent. The *accept* message is used to inform the agent about the link clicked with respect to a previously submitted query. The agent can respond with an *inform* message in case feedback is correctly processed, or with a *refuse* message in case the message is not valid or refers to the query which has not been processed by the agent.

Finally, for managing the history of previous searches, the system allows for the use of the User-Agent InsertLink Protocol (Figure 5.5(d)), which is similar to the User-Agent Feedback Protocol, but is used when the user inserts the link into the history of previous searches. The information about this fact is passed using the *insert* message with the parameters about the link, its type and the query. In the current implementation each time the user clicks on the link it is inserted in the personal history, but in principle, this protocol could be used in case of bookmarking the link.

---

[4]`http://www.fipa.org/specs/fipa00027/SC00027H.html`. FIPA Query Interaction Protocol Specification.
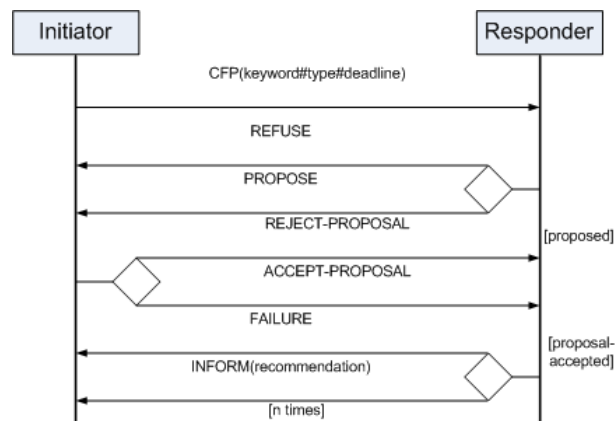
Figure 5.7: Agent Query Protocol. This protocol is used by personal agents to ask other agents for recommendations

Agent-agent interactions. Personal agents in Implicit interact with each other by exchanging messages in FIPA Agent Communication Language (ACL)[5] using several protocols. In the following we describe these protocols and the purposes for which they are used.

Figure 5.7 shows Agent Query Protocol that is used by personal agents when producing recommendations. It is a modified version of FIPA Contract Net Protocol[6]. In the FIPA Contract Net Protocol, the Initiator sends a *call for proposal (CFP)* message to all agents on the platform and then selects the best proposal. In our implementation when the Initiator sends a CFP message, it knows for sure who will be the Responder since it is the agent either discovered by the External search behavior or received as a referral from the DF or from other agents. Thus, the interaction starts with a CFP message that contains the type of the search (link or agent ID), the search keyword and the deadline for receiving a proposal. The Responder agent accepts the search in case it has enough resources for performing it[7]. If the Responder accepts the search it sends a *propose* message, otherwise it sends a *refuse* message. When the Initiator receives the propose message before the deadline specified in the initial CFP message, the Initiator sends a *accept-proposal* message to the Responder. However, if the propose message has been received after the deadline specified in the initial call for proposal, the Initiator sends a *reject-proposal* message. In the FIPA Contract Net Protocol this step is more complex because the Initiator must evaluate several proposals at this point. After the Responder finishes the search it sends an *inform* message that contains a recommendation. Such message can be repeated several times, in case the agent has produced more than one recommendation. Alternatively, the Responder can send *failure* message if for some reason it did not produce any results or cannot handle the search anymore.

Such interactions about recommendations are usually complemented with additional communication about the user feedback. Such interaction is directed from the Initiator of the Agent Query Protocol to the Responder and is handled by a separate Agent Feedback

---

[5]http://www.fipa.org/specs/fipa00061/SC00061G.html. FIPA ACL Message Structure Specification.

[6]http://www.fipa.org/specs/fipa00029/SC00029H.html. FIPA Contract Net Interaction Protocol Specification.

[7]To avoid agent overloading there is a predefined limit of agent searches one agent can proceed simultaneously.
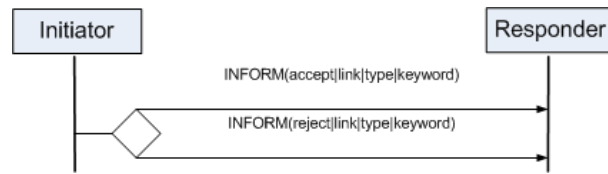
Figure 5.8: Agent Feedback Protocol. This protocol is used by personal agents to propagate user feedback to the agents who produced the recommendations.

Protocol, shown in Figure 5.8. The protocol consists of only one *inform* message sent from the Initiator to the Responder. This message contains information about which link of which type has been accepted or rejected for a query.

5.1.4   Related work

In this subsection, we review the research approaches related to the Implicit system. For convenience, we grouped the related work in several areas describing system for recommending contacts, community-based search engines, and agent-based approaches.

Recommending contacts. Vignollet et al. in [146] described a recommendation system that adopts the collaborative filtering and social networks analysis techniques. The system recommends contacts instead of contents. The idea behind contact recommendations is that users prefer others' advice to impersonal guidance and also appreciate enriching relationships with others. This system is similar to our work in the sense of taking into account the social aspect of the information search.

A multi-agent referral system MARS has been presented by Yu and Singh [155]. In that system, each user has a personal agent. The agents interact in order to provide users with answers to their questions. Agents are also able to give each other the links to the other agents, which is similar to recommending agent IDs in Implicit. There is a complex model of agent interactions in MARS. Each agent classifies the other agents as neighbors and acquaintances and their status in this classification determines the way of contacting them. The system uses ontologies to facilitate knowledge sharing among agents. The ontologies must be pre-defined and shared among all the agents, while we emphasize the facilitation of implicit knowledge sharing by managing documents, links and reference to people. Differently from our system, the agents in MARS do not answer all questions of other agents, but only those related to the interests of their users. The paper is focused more on general knowledge search rather than on web search. Finally, the system is mail-based while Implicit is a web-based system that adopts FIPA standards and uses JADE platform.

Community-based engines The Implicit system is related to community-based search engines, like I-Spy [130], Eurekster[8], and to social bookmarking services, such as Delicious[9]. However, the Implicit system differs from these systems in several aspects. First, Implicit Culture focuses more on an organizational community, rather than on an emergent or online one. Second, it uses collaboration and interactions among agents to im-

---

[8]Eurekster. `http://www.eurekster.com/`
[9]Delicious. `http://delicious.com/`

prove suggestions. Third, it recommends also agents therefore establishing implicit trust relationship in the community. Finally, our system can be used to filter and re-arrange the results from systems such as Delicious to a specific user community.

Agent-based systems for improving web search. Menczer [100] suggests complementing search engines with online web mining in order to take into account the dynamic structure of the web and to recommend recent web pages which are not yet known by common search engines. To achieve this goal the adaptive population of web search agents united in the multi-agent system emulates user browsing behavior. The system consists of InfoSpiders, which are the agents incorporating neural net and analyzing the links and the context of the documents corresponding to the links on the current page in order to propose new documents to the user. The main goal of this system is the discovery of new information, not yet presented in web search engines, in order to provide more up-to-date service to the user.

A collaborative multi-agent web mining system called Collaborative Spiders was developed by Chau et. al [35]. The system implements the post-retrieval analysis and enables across-user collaboration in web search. In order to provide a user with recommendations a special agent performs profile matching to find the information potentially interesting to the user. Before the search, the user has to specify the area of the interest and privacy or publicity of the search. Unlike to Implicit, in the Collaborative Spiders system users should analyze excessive system output because they have to browse through a number of similar already finished search sessions.

SurfAgent [131] is an information agent that builds a user profile by using user-supplied examples of relevant document. The authors presented and evaluated the mechanism of automatic query generation from the user profile and using the generated queries to provide relevant documents to the user. Such approach of pro-active searching for documents that might be interesting for the user is called the "push" approach. Implicit applies the "pull" approach when recommendations are delivered to the users only when they search. Also, we do not represent information about user searches explicitly in user profiles. Therefore, query generation from user profiles is not applicable in our system.

Other related work. The Implicit system can be used for supporting collaboration of the community members and sharing experience about using particular web links relevant to their specific interests. In this regard Implicit is complementary to the work by Geczy et al. [57] who investigated patterns in browsing behavior of a community of knowledge workers.

A recommendation model presented in [147] produces recommendations by using the social network existing between users and modeling the trust relationships with neighbors. The topic of using trust in recommendation systems is deeply investigated in papers by Massa (see, for instance, [96]). Differently to such systems, in Implicit we do not model the social network and trust relationships explicitly. However, trust relations and social ties emerge from interactions between agents. In the conducted simulations we noted that after a certain number of queries, the SICS of each agent mainly contacted only one single agent, who gave the most relevant recommendations in the past.

5.2   Software pattern selection

Almost fifteen years ago the authors of the book *Design Patterns* [56], the first major publication on software patterns, stated the problem of selecting patterns: "With more than 20 design patterns in the catalog to choose from, it might be hard to find the one that addresses a particular design problem, especially if the catalog is new and unfamiliar to you". As time has passed, patterns have become a staple of current software development approaches. However, the problem of selecting patterns still exists. Moreover, it has become much more critical as the number of documented patterns is continuously increasing: for instance, the *Pattern Almanac* [116] lists more than 1200 patterns. And in the past nine years since its publication, many new patterns and books on patterns have been published. The problem of choosing the appropriate pattern is particularly hard to solve for inexperienced programmers [132], and tools assisting in this process become of utmost importance [122].

Although the problem of pattern selection can be considered as a particular instance of the general problem of retrieval of relevant information from large document collections [50], it requires specific tools, due to a number of reasons: (i) patterns are structured documents where different parts express extremely different information; (ii) they are often linked to each other in a pattern language; (iii) patterns accumulate the experience of developers in dealing with design problems. Therefore, besides pattern catalogs [65], existing approaches for supporting pattern selection include case tools [59], expert systems [86], and formal frameworks that help to reuse knowledge about patterns [66, 156]. However, the existing approaches that support the developers in the selection of patterns do not take into account social factors, collaboration and personalization.

In this section, we present the IC-Patterns system that support users in the process of making decision about which pattern to use for their design problem. The system addresses the problem of pattern selection from a social point of view. To help a developer make a decision about which patterns to use, getting suggestions from her group of peers is important. The system supports collaboration among users by using the Implicit Culture approach that allows developers to share knowledge about the patterns they use for various design problems. The multi-agent architecture facilitates such knowledge sharing because personal agents in the system allow for sharing knowledge about the use of patterns in a community of developers without their direct involvement. Namely, agents provide their users with suggestions on which patterns are suitable for a specified problem. The suggestions are complemented with a description of patterns from the pattern repository retrieved using IR and CBR methods.

5.2.1   Software patterns

Software patterns enable an efficient transfer of design experience by documenting common solutions to recurring design problems in a specific context [3]. They contain valuable knowledge that can be reused by others, in particular, by less experienced developers. Each pattern describes the situation when the pattern can be applied in its context. The context can be thought of as a precondition for the pattern. This precondition is further refined in the problem description with its elaboration of the forces, i.e. design trade-offs

affected by the pattern, that push and pull the system to which the pattern is applied in different directions. Here, the problem is a precise statement of the design issue to be solved. One of the most significant contributions of patterns is that they intend to make the trade-offs between the forces involved explicit. The trade-offs can be documented in various forms. One popular approach is to document them as sentences like "on the one hand ..., but on the other ...". The solution describes a way of resolving the forces. Some forces may not be resolved by a single pattern. In this case, a pattern often includes references to other patterns, which help resolve forces that were unresolved by the current pattern. Together, patterns connected in this way are often referred to as a pattern language. Links between patterns can be of different types, including uses, refines, and conflicts [105, 119]. Patterns that need another pattern link to that pattern with its uses. Patterns specializing the context or problem of another pattern refine it. Patterns that offer alternative solutions conflict, and should not be used together.

Patterns have been published for system architecture and detailed design, as well as for specific application domains (e.g. agents [5, 80, 43] and security [53, 67]). Recently, there have been several efforts in making patterns available in online pattern repositories, where they can be browsed and searched by various criteria. An early example was the Pattern Almanac [116], which is available in electronic form (`www.smallmemory.com/almanac`). More recent examples are the `patternshare.org` site hosted by Microsoft in 2006-2007, Yahoo Design Pattern Library[10], Sun collection of J2EE patterns[11], computer-mediated interaction patterns[12]. These catalogs rarely contain personalized features, although they can provide customizable pattern properties for enhancing search [65]. In order to store patterns in a repository, a structured pattern representation must be adopted. There have been several proposals for structural pattern representation, most notably the Pattern Language Markup Language (PLML) [54].

Our motivation for adopting an Implicit Culture approach in the system for choosing software patterns stems from: (1) the continuous increase in the number of documented patterns, for instance, the *Pattern Almanac* [116] lists more than 1200 patterns; (2) the difficulty less experienced developers face in using patterns. Developers who wish to apply patterns from a domain that is not their main area of expertise encounter similar difficulties. A good example is the security domain. For any but trivial applications, security is a key concern, however, making the application secure is not the main concern of the application developer. What a developer wants is to be able to focus on the core application functionality. Security patterns [119] can help developers with the task of adding security into an application: they provide guidance to non-experts in security for designing secure application. However, a significant challenge remains: how do developers decide which patterns they should use?

The following quote from Sommerville [132] is indicative of the difficulty inherent in using patterns:

> Only experienced software engineers who have a deep knowledge of patterns can use them effectively. These developers can recognize generic situations where a pattern can be applied. Inexperienced programmers, even if they have read

---

[10]`http://developer.yahoo.com/ypatterns/`
[11]`http://java.sun.com/blueprints/patterns/`
[12]`http://www.cmi-patterns.org/`

action	objects(attributes)
request	problem_description(keywords), project_description(ProjectName, SecurityLevel, ProjectSize)
apply	pattern(PatternName), problem_description(keywords), project_description(ProjectName, SecurityLevel, ProjectSize)
reject	pattern(PatternName), problem_description(keywords), project_description(ProjectName, SecurityLevel, ProjectSize)

Table 5.3: Mapping the pattern selection domain to the terms of the Implicit Culture Framework.

the pattern books, will always find it hard to decide whether they can reuse a pattern or need to develop a special-purpose solution.

The difference between these two types of developers is that an experienced developer uses implicit knowledge (in particular, her own experience) about the problem (see [45] for a more general discussion on this point).

We argue that it is possible to shift the pattern selection behavior exhibited by inexperienced developers towards the behavior of more experienced developers by suggesting patterns suitable for their current design task. To determine which patterns are suitable we use the history of previous user interactions with the system, i.e. which patterns other developers have chosen in similar situations.

5.2.2   Applying the Implicit Culture Framework

We refer to the pattern selection behavior of experienced developers as the *culture* of that developer community. When inexperienced developers start behaving in agreement with the community culture, behavior transfer from experienced to inexperienced developers occurs and the system . The relation characterized by this transfer is the implicit culture relation.

This section explains how to apply the Implicit Culture approach to the problem of pattern selection and presents the IC-Patterns system that helps to choose software patterns and describes the retrieval process within the system. The system is intended for the use within an IT-company, or just within a project group, and it should adapt the suggestions on the use of software patterns to the specificity of the software development process adopted within the company or project group, converging to the "community culture".

In the context of the problem of pattern selection the Implicit Culture approach consists in (i) observing how developers search for patterns and which patterns they select among those proposed and (ii) recommending developers patterns applied for similar problems in the past. The similarity of the problems is defined as the similarity between the submitted queries.

In terms of the Implicit Culture Framework developers are *agents*, while patterns, problem descriptions and project descriptions are *objects*. Table 5.3 summarizes actions and objects in the pattern selection domain. Since all the actions are performed by developers, we omit agents from the table.

We explain the information contained in the table in detail. A developer *requests* the system to find patterns that are suitable for her task. The query contains a *description of the problem* and a *description of the project* where the problem has been encountered. The problem description includes the attribute *keywords* which contains the keywords of the query, while the description of the project contains the attributes *ProjectName*, *SecurityLevel*, and *ProjectSize*.

The developer *applies* the pattern, identified with the attribute *PatternName*, when she implements it in the code, and can specify the inapplicability of a pattern to the task as a *reject* action.

Example. Let us consider a repository of security patterns and a programmer who needs to improve access control in a system that offers multiple services. Let us suppose that for an experienced developer knowledgeable in security it is apparent to use the Single Access Point pattern (Figure 5.11). If the system is able to use previous history to suggest that the novice uses the Single Access Point pattern and she actually uses it, then we say that she behaves in accordance with community culture and the implicit culture relation is established. We use this example as a running example throughout the section.

In this example, actions are: `request(query)`, `apply(SingleAccessPoint, query)`, `reject(Authenticator, query)`, where `query` contains `problem_description` and `project_description`.

In terms of our problem domain, the observer module of the SICS stores the submitted query, which patterns have been proposed as a solution, and which pattern has been chosen in return. We do not use the inductive module of the SICS in the system, so the following theory that consists of one rule is pre-defined:

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

This means that the apply (and not, e.g. a *reject*) action must follow the *request* action. The composer module of the SICS tries to match the problem expressed by the query with the pattern by analyzing the history of observations and calculating the similarity between the problem description given by the user and the problem descriptions which users provided previously. The patterns selected for latter descriptions, i.e the patterns previously selected for similar problems, are recommended.

Obviously, the main problem lies in the "observability" of the users' actions. The most problematic action to observe is the action of using a pattern for a problem. In the current implementation we assume the user explicitly indicates this action in the system, specifying that she selected the pattern X for the problem A, where the problem corresponds to a search in the history of searches. This is a reasonable assumption, since the amount of the input required from the user is very low.

5.2.3 The IC-Patterns system

The architecture of the system is given in Figure 5.9. The system consists of a web-based user interface on the client side and a multi-agent platform on the server side. A user accesses the system by submitting a query via the web-based interface in her browser. In the IC-Patterns system a query includes a description of the problem and a description of the project where the problem is encountered. The problem is described

by a set of keywords, optionally restricted to specific elements of the pattern description, e.g. problem, context. The project description can be represented as a set of properties such as project size, required level of data protection, etc.

Example. In our running example, the user could submit a query with the following problem description: "complex security control" related to the project that has the following set of properties: {Name: OnlineBanking, SecurityLevel: High, ProjectSize: Medium}. The other considered projects have the following properties: {Name: e-BookShop, SecurityLevel: Medium, ProjectSize: Medium}, {Name: eLections, SecurityLevel: High, ProjectSize: Big}.

Each user is assisted by a personal agent. The goal of the personal agent is to help the user to choose a pattern suitable for the submitted query. In order to fulfill this goal, the agent can access the SICS via the IC-Service (the BQICS module), and to access the repository of patterns directly via the Information Retrieval (IR) API provided by Lucene (BQR-IR) and using the Case-Based Reasoning (CBR) module (BQR-CBR). The personal agents in the system are software agents running on the multi-agent platform. The IC-Service in the system is used in order to recommend patterns, namely it provides an interface for accessing a SICS that is dealing with observations coming from the system and produces recommendations. The user query is treated in different way by the three agent modules. When querying the repository using the IR methods, only the keywords are used and they are compared with free-text descriptions of patterns in the repository. If the repository is accessed using CBR methods, the keywords are treated as a description of the problem, and is compared with the "Context" and the "Problem" sections of the pattern descriptions. When querying the IC-Service, both keyword and project parts of the query are used.

Example. The user's personal agent should suggest using the Single Access Point pattern. If the agent does so because someone else has already used this pattern for similar problems, it transfers the knowledge about the use of patterns within the community.

Multi-agent architectures have been already used in decision support systems [27] and such architecture provide a number of benefits. First, agents provide users with personal assistance, i.e. each agent personalizes the system to its user. Second, agents provide an interface to different recommendation mechanisms without the need of heavy client part of the system. Third, agents allow for the use of the system in distributed settings. Finally, in the simulations we conducted to validate the system, each agent contained a model of the user in order to simulate users of the system. Overall, the use of agents provides a flexible and implicit way of sharing information about actions: they use the IC-Service to answer user queries about cases, provide retrieved cases, and store all the actions in the database of observations. Agents can also interact with one another to share expertise and knowledge of their users in using the case base.

The use of the Implicit Culture approach for recommendations allow for sharing of the knowledge about the use of patterns without the direct involvement of the users. The IR and CBR recommendation mechanism allow to overcome the cold start problem [28], i.e. inability to suggest items in the beginning of the use of the system. The combination of IR and CBR methods allow for getting more recommendations, since the results returned by the two methods are, in general, different.
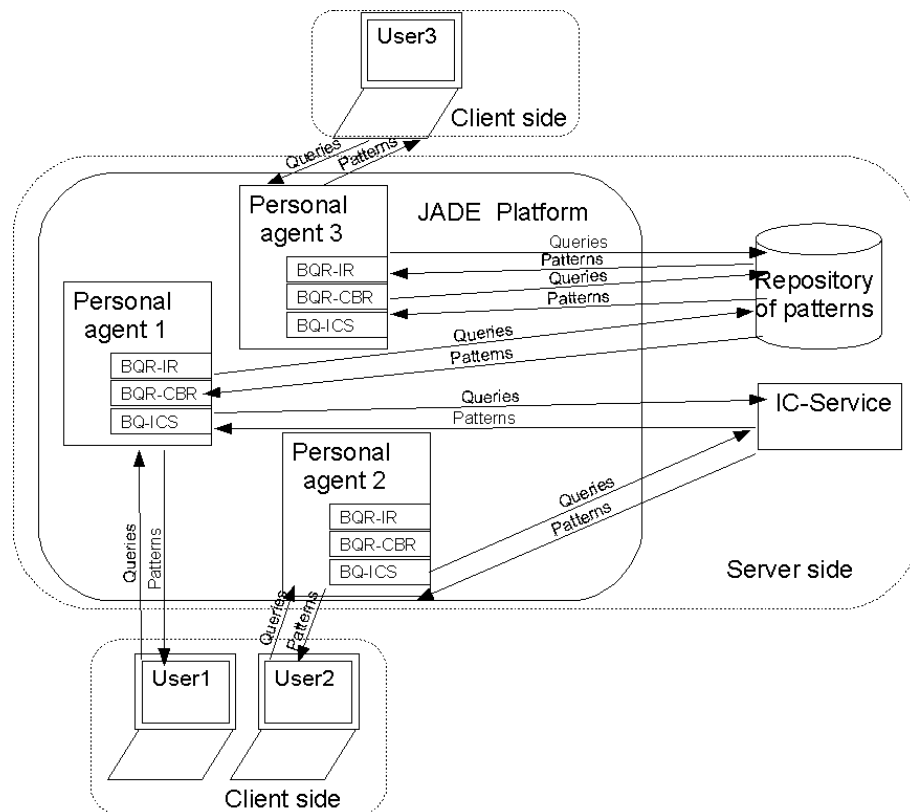
Figure 5.9: The architecture of the system. *Personal agents* process queries from *users* and retrieve potentially relevant patterns from the repository of patterns; the IC-Service is exploited by the agents in order to create recommendations from the history of past interactions; *BQR-IR* stands for BehaviourQueryRepository-IR used to access the repository using the IR methods, *BQR-CBR* stands for BehaviourQueryRepository-CBR used to access the repository using the CBR methods, and *BQ-ICS* stands for BehaviourQueryICService, respectively.
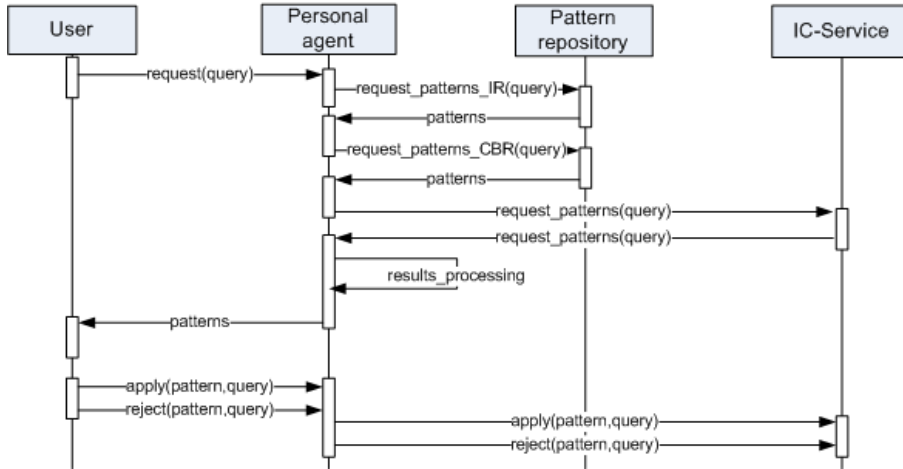
Figure 5.10: Sequence diagram of the search process.

Search in the system

The sequence diagram of the search process is given in Figure 5.10. A user submits a query via the user interface, from where the query is forwarded to the user's personal agent. In the first step of the search process, the personal agent accesses the pattern repository using IR methods and retrieves a set of patterns relevant to the query. In the second step, the repository is accessed using CBR methods. In the third step, the personal agent submits a query to the IC-Service and receives a list of recommended patterns. The list is processed by the agent, e.g. patterns are ranked and duplicates are removed. Thus, the results contain patterns retrieved from the repository by IR and CBR methods and patterns recommended by the IC-Service. As the last step the feedback of the user is collected via the *apply* and *reject* actions.

The SICS inside the IC-Service processes the query within two steps. In the first step, the SICS matches the action contained in the query, i.e. the *request* action, with the theory and determines the action that must follow, i.e. the *apply* action. During this step the SICS also fills in the parameters of the *apply* action, for instance, the *problem_description* object. In the second step, the SICS finds situations where the *apply* action with similar parameters has been previously performed, thus determining the patterns used for similar problems in the past. Since *problem_description* is a part of the *apply* action, the similarity between the current query and the previously submitted queries is calculated. As the result, the SICS returns a set of patterns that have been used for similar problems in the past.

Example. Let us illustrate how the search process takes place in our example. The user submits the *request* action with the following query: {ProblemDescription: "complex security control"; Project: {Name: OnlineBanking, SecurityLevel: High, ProjectSize: Medium}}. In the first step the agent retrieves patterns from the repository: SingleAccess-Point and RoleBasedAccessControl. In the second step, the agent queries the IC-Service. The SICS matches the *request* action with the left part of the theory that represents a problem, and searches for situations where the *apply* action has been performed. It finds

the following situations (situation_id, the action, problem description, project, pattern):

1	apply	access control in a system that offers multiple services	pp	SingleAccessPoint
2	apply	only authorized clients should access the system	pp	PolicyEnforcementPoint

where pp={Name: e-BookShop, SecurityLevel: Medium, ProjectSize: Medium}. As a result, the SICS returns the SingleAccessPoint pattern, chosen in the most similar situation w.r.t. the submitted query[13]. After the evaluation of the results, the following list of patterns is displayed in the user interface: {SingleAccessPoint, PolicyEnforcementPoint, RoleBasedAccessControl}. Having analyzed the proposed patterns, the user *applies* the SingleAccessPoint pattern and indicates this in the user interface. She also marks the RoleBasedAccessControl pattern as unsuitable, thus performing the *reject* action.

Implementation details

The system is implemented using JADE 3.4.1 (Java Agent DEvelopment framework) and uses the IC-Service for the retrieval of patterns. It uses the IC-Service as a Java library.

For the repository of patterns we have adopted a format that is specific to a set of security patterns previously hosted at `patternshare.org` [67]. We have defined an XML representation for these patterns and extracted the content of the subset of this repository from the website. Our current representation contains the following elements: Pattern.Name, Pattern.Context, Pattern.Problem, Pattern.Solution, and Pattern.RelatedPatterns, as well as elements specific to the `patternshare.org` site, but not required for our purposes (see Figure 5.11 for an example of the pattern representation). However, our approach does not depend on a specific pattern representation. Note, that although the representation of a pattern in this system is a structured one, when accessing the repository with IR methods, it is treated as a free-text representation.

We are also not concerned, at this stage of development, with how easy it is to deploy our approach for building a repository; however, in the future; we plan to converge towards a standard, like PLML, for pattern description.

To build the repository of patterns we took the following steps as shown in Figure 5.12: (1) the descriptions of security patterns were extracted from `patternshare.org` using scripts; (2) the pattern descriptions were converted to the XML format using scripts; (3) the XML documents representing patterns were indexed with Apache Lucene 2.4. The Lucene library is also used by personal agents to access the repository of patterns. However, our approach does not depend on a particular repository or a tool for accessing the repository. Moreover, the repository can be further extended with adding other patterns and pattern collections.

Since Apache Lucene provides the opportunity to have different weights for different sections of the documents, we have used Lucene also for performing CBR. For this task, the "Problem" and "Context" sections of a pattern have weight equal to one, while the other sections have zero weight, i.e. removed from the similarity calculation in the retrieval

---

[13]Without going in detail of the general algorithm of similarity calculation, let us say that the similarity between two actions in this case is calculated based on the similarity of names of actions and objects. In this case we have two objects: problemDescription and projectDescription, and the similarity between problem descriptions is calculated as the fraction of common terms, while the similarity between project descriptions is calculated as the fraction of equal properties (ProjectName,ProjectSize, SecurityLevel).

```
<Pattern id="SingleAccessPoint">
<Pattern.Name>Single Access Point</Pattern.Name>
<Pattern.View>Application Architecture</Pattern.View>
<Pattern.Role>Architecture</Pattern.Role>
<Pattern.Aspect>Function</Pattern.Aspect>
<Pattern.Summary>Single entry point for each process.</Pattern.Summary>
<Pattern.Context>You are planning to secure a system from outside intrusion. The system
  provides a bunch of services but you want to secure the system as a whole.
  </Pattern.Context>
<Pattern.Problem>A security model is difficult to validate when there are multiple ways
  for entering the application. How can we secure a system from outside intrusion?
  </Pattern.Problem>
<Pattern.Solution>Set up only one way to get into the system and if necessary, create a
  mechanism to decide which sub-application to launch. Typically most applications use
  a log in screen to accomplish the single access point. </Pattern.Solution>
<Pattern.RelatedPatterns>Single Access Point validates the user's login information
  through a <Pattern idref="PolicyEnforcementPoint"/> and uses that information to
  initialize the user's Roles and Session. A Singleton can be used to implement a Single
  Access Point. </Pattern.RelatedPatterns>
<Pattern.Publication>This pattern appeared in the paper titled "Architectural Patterns
  for Enabling Application Security" by Joseph Yoder and Jeffrey Barcalow in Pattern
  Languages of Programs conference in 1997. Peter Sommerlad integrated the material in
  the Security Pattern book titled "Security Patterns: Integrating Security and Systems
  Engineering". </Pattern.Publication>
</Pattern>
```

Figure 5.11: An example of the XML representation of the Single Access Point pattern in our pattern markup language.

Figure 5.12: The pattern extraction process. (1) Information about the security patterns is extracted from the patternshare.org repository using Perl scripts; (2) the pattern descriptions are then converted to an XML format using Perl scripts and a pattern markup language.

process.

5.2.4 Related work

In this subsection, we review the research approaches related to the IC-Patterns system.

PatternSeer [112], is an ongoing project that aims at delivering a system that crawls and indexes pattern descriptions on the Internet and makes them accessible for the users via keyword-based search. Recently, Google provided a custom search engine[14] indexing several online pattern repositories.

There are several approaches that use CBR for the retrieval or recommendation of software pattern. For instance, a system for the retrieval of semantic templates for designing recommender systems [77] and the system for the reuse of software examplets [64]. The ReBuilder framework [59] adopts a CBR approach [1], where cases represent situations (problems) in which a pattern was applied in the past to a software design. ReBuilder supports the retrieval and adaptation of patterns. Cases are described in terms of class diagrams. Cases are retrieved based on a combination of structural similarity between the current design and a pattern, as well as the semantic distance between class names and role names in the pattern. Our approach is complementary to the one used in ReBuilder as patterns are selected on the base of previous actions of other users. Also, while the use of the relations in the class diagram provides additional information about the desired pattern, such diagrams are not always available. However, the textual descriptions of patterns are always available, and since our system uses the textual descriptions, it has a wider range of potential applications, although, probably, it can not compete with ReBuilder in the domains where class diagrams are available. Finally, the IC-Patterns system implements a collaborative approach to pattern selection, because the Implicit Culture Framework facilitates experience sharing among the users.

Kung et al. [86] propose a methodology for constructing expert systems for suggesting design patterns to solve problems faced by developers. They present a prototype, the Expert System for Suggesting Design Patterns (ESSDP), which implements the methodology. ESSDP selects a design pattern based on the user's requirements. A user interacts with the system using question-answering approach, which helps to narrow down the selection process. At the end of the interaction, a suitable design pattern is offered to the user. There are several significant differences between our approach and the ESSDP system. First, ESSDP assumes the knowledge acquisition as the primary step of the methodology. In this step human experts must fill in the knowledge base with some pre-defined rules. Differently, in our system the suggestions come from the interactions with users, without any initial knowledge base, allowing for continuous improvement of suggestions. Moreover, we exploit interactions with inexperienced users as well, offering to novices patterns that have been chosen in similar situations not only by experts but also by other novices. Thus we support sharing users' experience with others. Second, our system is not restricted to the use of a rule-based knowledge base assuming that different learning techniques can be adopted.

Several approaches propose adding formal semantics into pattern descriptions. For instance, Gross and Yu [66] present a formal approach, proposing to add non-functional

---

[14]Design pattern search. `http://www.google.com/coop/cse?cx=000531763273211731096%3Ab-lv61obcte`

requirements into descriptions of patterns and use such requirements for the retrieval. Similarly, Wang et al. [148] use non-functional requirements framework to retrieve patterns that might be suitable for a given set of requirements and will result into a detailed design. Many patterns were not developed individually, but rather they were organized in pattern languages. Some approaches target the selection of pattern(s) from such languages thus handling relations between patterns, not only individual pattern descriptions. Zdun [156] proposes an approach for pattern selection based on desired quality attributes. The approach requires formalizing the pattern relationships in a pattern language grammar and annotation of the patterns with effects on quality goals. As a result, the search space is narrowed down and the time spent evaluating alternatives is decreased. Mussbacher et al. [104] present goal-oriented requirement languages that formalizes forces of patterns and relations between patterns.

Most of the existing approaches require manual interventions in the process, such as specifying additional information about patterns or their relations, creating a knowledge base, or organizing the collection in a specific way. In contrary, our system can handle any repository of pattern and provide recommendations using one of the three techniques or their combination. Moreover, due to the multi-agent architecture and the architecture of the IC-Service recommendation engine used in the system it is possible to use the system in distributed settings, e.g. in different branches of a company.

5.3   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 [94], and the use of syntactic, semantic and structural reviews of web service specifications [79]. 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 with 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.

To support such information exchange, the idea of applying recommendation systems for discovering and selecting web services has been recently proposed [22, 81, 93, 126]. Existing recommendation-based approaches use ratings of service providers based on explicit and often subjective opinions of service clients [126]. However, as demonstrated in [37], 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 participate personally in 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. On the developers side, the effort requested 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. This kind of information can be particularly useful for dynamically reconfigurable systems to support self-healing behavior.

In this section, we present an implemented system for improving web service discovery. The system is based on the IC-Service described in Chapter 4. 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 is 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[15] lexicon.

5.3.1 Applying the Implicit Culture Framework

With respect to the meta-model of the Implicit Culture terms (Section 4.2) in our application *agents* are developers who submit requests for web service operations represented as *objects*. Names of web services and information about their providers are stored as *attributes* of operations, while submission of requests, service invocations and corresponding responses are modeled as *actions*. An example of a *scene* could be a set of actions corresponding to the invocations of various service operations:

$$invoke(\text{getWeatherByZip } (service = \text{DOTSFastWeather});) \text{ or}$$
$$invoke(\text{getWeather } (service = \text{GlobalWeather});).$$

An example of a performed action could be

$$invoke(\text{Peter; getWeatherByZip } (service = \text{DOTSFastWeather});; \text{25-Jun-07-14:22})$$

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.

In this application the SICS is deployed as a web service and assessed via the SICS Remote Client (Figure 5.13). The cultural theory for web service discovery contains the following rule:

$$\text{if } submit\_request(\text{request-X}) \text{ then } invoke(\text{operation-Y(service-Z), request-X}).$$

This means that the *invoke* action must follow the *submit_request* action and both actions are related to the same request.
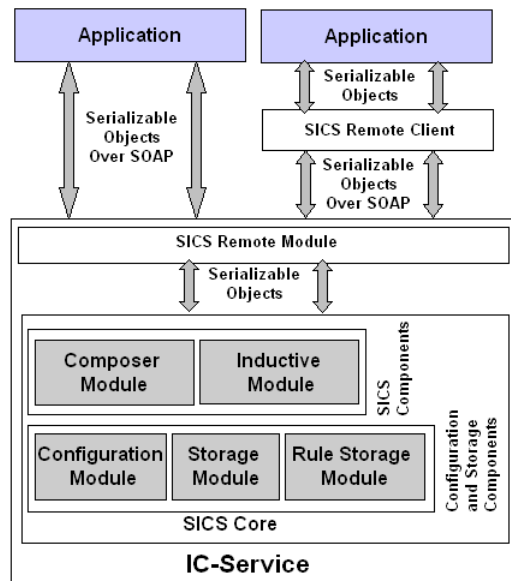
---

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

Figure 5.13: The general SICS architecture. The `Composer Module` provides recommendations facilities; the `Inductive Module` discovers a theory that expresses the community culture; all parameters of a SICS instance are configured in the `Configuration Module`; the `Storage Module` is responsible for storing the information about the application domain (agents, actions, observations, etc.); the `Rule Storage Module` is responsible for the management of the theory such as adding or removing theory rules.

5.3.2 The system for web service discovery

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 5.14). To join a community that shares experience about service usage, developers must include into their application the SICS Remote Client that enables monitoring of web service invocations on the client side.

The working scenario is as follows: an agent submits a request to the IC-Service, which returns a list of recommended services. The request contains a textual description of the goal, the name of the desired operation, the description of its input and output parameters, the description of a desired web service and an optional list of preferred features (provider, etc.). It is stored in the system as 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.

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
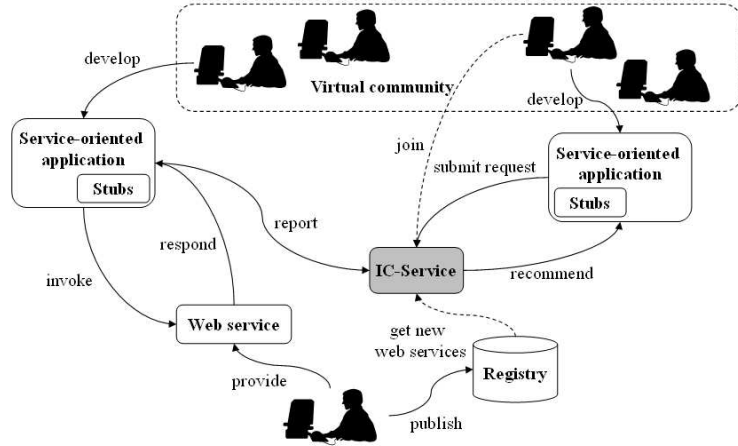
Figure 5.14: Web service monitoring and discovery with IC-Service.

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.

Similarity configuration

Similarity between observed actions (such as *submit_request*, *invoke*, etc.) is determined by the similarity of their names, attributes and objects. The main element taken into account in our system is the request represented as a sequence of terms $q = (t_1, t_2, ..., t_{|q|})$, where $|q|$ is the length of the request and $t_j \in T, j = \overline{1, |q|}$. T is a vocabulary of terms, containing all terms from the collection of requests $Q = \{q_1, ..., q_n\}$ submitted to the system, where n is the total number of requests. We use two different similarity metrics in the composer module to calculate the similarity between requests: the Vector Space Model (VSM) with Term Frequency - Inverse Document Frequency (TF-IDF) metric and a WordNet-based semantic similarity metric.

For calculating the first metric, for each term t_j let n_{ij} denote the number of occurrences of t_j in q_i, and n_j the number of the requests that contain t_j at least once. The TF-IDF weight of the term t_j in the request q_i can be obtained as follows:

$$w_{ij} = \frac{n_{ij}}{|q_i|} * \log(\frac{n}{n_j}),$$

where $|q_i|$ defines the length of the request q_i. The similarity between requests q_i and q_k is defined using the cosine coefficient:

$$sim(q_i, q_k) = cos(w_i, w_k) = \frac{w_i{}^T w_k}{\sqrt{w_i{}^T w_i}\sqrt{w_k{}^T w_k}}.$$

Here $w_i = (w_{i1}, ..., w_{i|T|}), w_k = (w_{k1}, ..., w_{k|T|})$ denote vectors of TF-IDF weights corresponding to the requests q_i and q_k, and $|T|$ is the length of the vocabulary T.

In the second metric, for defining a similarity between requests $q_i = (t_{i1}, t_{i2}, ..., t_{i|q_i|})$ and $q_k = (t_{k1}, t_{k2}, ..., t_{k|q_k|})$, we first calculate the lexical similarity between any pair of their terms $t_{ij}, j = \overline{1, |q_i|}$ and $t_{kl}, l = \overline{1, |q_k|}$. For defining a lexical similarity of all possible senses of two terms we use a WordNet-based metric designed by Seco et al. [121]:

$$sim(t_{ij}, t_{kl}) = 1 - \frac{1}{2}(ic_{wn}(t_{ij}) + ic_{wn}(t_{kl}) - 2sim_{res'}(t_{ij}, t_{kl})),$$

where

$$sim_{res'}(t_{ij}, t_{kl}) = \max_{t \in S(t_{ij}, t_{kl})} ic_{ws}(t).$$

In this expression, $S(t_{ij}, t_{kl})$ denotes the set of concepts that subsume t_{ij} and t_{kl}. Information context value of a WordNet concept is defined as

$$ic_{wn}(t) = 1 - \frac{log(hypo(t) + 1)}{log(max_{wn})},$$

where *hypo* is the number of *hyponyms*, i.e., words whose extension is included within that of another word of a given concept, and max_{wn} is the maximum number of existing concepts. We extended this metric to deal with sets of words. The problem of calculating the overall similarity between requests q_i and q_k is formulated as the Maximum Weight Bipartite Matching problem in a complete bipartite graph, where terms from q_i and q_k define two partitions of vertices and the obtained lexical similarity values $sim(t_{ij}, t_{kl})$ define weights of edges. The goal is to find a *matching M* (i.e., a subset of edges $e_{jl} = (t_{ij}, t_{kl})$ such that no two edges in M share a vertex) with the maximum total weight. This weight defines the similarity between requests q_i and q_k, i.e.,

$$sim(q_i, q_k) = \max_M \sum_{j=1}^{|q_i|} \sum_{l=1}^{|q_k|} \sigma_{jl} \, sim(t_{ij}, t_{kl}),$$

where

$$\sigma_{jl} = \begin{cases} 1, \text{ if edge } e_{jl} = (t_{ij}, t_{kl}) \in M, \\ 0, \text{ otherwise.} \end{cases}$$

Example of search in the system

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

$$\begin{array}{rl} goal : & \text{get weather forecast for Rome, Italy;} \\ operation : & \text{get weather;} \\ input : & \text{city name, country name;} \\ output : & \text{weather forecast (temperature, humidity, etc.).} \end{array}$$

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 scenes:

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

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

3. *invoke*(...; getWeatherByZip (*service* = DOTSFastWeather), *goal* = return 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. The observed *invoke* action can be used in future queries as a feedback about the service.

5.4 Concluding remarks

In this chapter, we described three applications of the Implicit Culture Framework: a system for recommending web links, a system for recommending software patterns, and a system for web service discovery. All three applications can be classified as recommendation systems, but they differ in the domain of recommendations and in the implementation. The differences in the domains are summarized in Table 5.4 and explained below, while the differences in the architectures are summarized in Table 5.5.

The Implicit and the IC-Patterns systems implement multi-agent architectures with personal agents helping their users, while the system for web service discovery implements a service oriented architecture. In the Implicit system, each personal agent has a SICS, and, therefore, does not depend much on others in creating recommendations. Differently, in the IC-Patterns system, there is only one SICS on the multi-agent platform, because in that scenario, it is assumed that community of the users is more homogeneous. In the web service discovery scenario there is one SICS used for each community of developers. Moreover, Implicit uses SICS both for recommending web links and contacts, i.e. agents in the community who are considered as knowledgeable in the topic.

application	agents	actions	objects(attributes)	cultural theory
Implicit	people searching the Web	request, accept, reject	link, query(keyword)	**if** *search*(agent;query) **then** *accept*(agent;link,query)
IC-Patterns	developers	request, apply, reject	pattern(PatternName), project_descr(ProjectName, ProjectSize, SecurityLevel), problem_descr(keywords)	**if** *request(agent;query)* **then** *apply*(agent;pattern,query)
WS-Discovery	developers	submit_request, invoke, get_response, raise_exception	request(Goal, OperationName, Input, Output), WSOperation(WSName, WSProvider)	**if** *submit_request*(agent;request) **then** *invoke*(agent;WSOperation,request)

Table 5.4: The comparison of the application domains.

application	architecture	using SICS	using IC-Service
Implicit	MAS	in each agent, recommending links and contacts	N/A
IC-Patterns	MAS	one for all, recommending patterns	Java library
WS-Discovery	SOA	one for all, recommending service operations	remote client

Table 5.5: The comparison of the application architectures.

The applications access the SICS in different ways. Implicit was developed earlier than the IC-Service, so it uses an ad-hoc implementation of the SICS architecture as a Java library. The IC-Patterns system uses the IC-Service as a Java library, while the system for web service discovery uses the IC-Service as a web service via the SICS Remote Client.

Finally, the domains of the three applications differ in terms of actions, objects, attributes and cultural theory. However, in spite of the differences of the applications, the Implicit Culture Framework proved to be useful in all systems, as shown in the next chapter, which presents a methodology and the results of the evaluation of the systems.

Chapter 6

Experimental evaluation

In this chapter, we present the objectives, methodology, and results of the evaluation of the systems presented in the previous chapter.

Section 6.1 presents objectives of the evaluation and provides an overview of the methodology we applied for the evaluation of the developed applications. Thorough evaluation of a system with real users is hard because it requires a lot of time from the users. Therefore, we conducted simulations using the developed systems. The simulations provide a way for an extensive evaluation of the systems. We present the description of the experiments and the results of the quality and performance evaluation of the systems in Section 6.2. The results of a real-user study carried out using the Implicit system are provided in Section 6.3. Section 6.4 describes the scalability experiments we conducted using the IC-Patterns system, and Section 6.5 discuss the results of the evaluation.

6.1 Objectives and the evaluation methodology

This section presents objectives and the methodology of the evaluation. Subsection 6.1.1 describes the objectives. For evaluation we simulated real users using the user model described in subsection 6.1.2. Subsection 6.1.3 describes the scalability dimensions of the systems and the performance and quality measures for evaluating the results.

6.1.1 Objectives of the evaluation

We identified the following objectives of the evaluation:

- To test the performance of the systems, in terms of the response time.

- To test the retrieval performance, i.e. the quality of results, of the systems, and compare the retrieval performance of different algorithms.

- To measure the impact of the SICS in each system by comparing the quality of suggestions with and without the SICS

- To test the scalability of the systems.

application	query	user profile	acceptance strategy
Web search	one keyword	links and click-through rate	a link is accepted if its click-through rate is greater than or equal to some relevance threshold
Pattern selection	set of keywords	relevant patterns	a pattern is accepted if it is in the profile
WS discovery	brief description of an operation	operations for groups of requests	an operation is accepted for a submitted request if it is in the profile for the request

Table 6.1: User models used in different application scenarios.

- To test the systems with real users to see how the systems are accepted by people and whether they provides enough support for sharing experience in a community of practice.

6.1.2   The user model

To simulate user behavior in applications described in Chapter 5 we developed a user model. The main functions of the user model were: (1) provide pseudo-user input in order to enable recommendations, and (2) generate pseudo-user response to the recommendations. Thus, the user model contained two parts that represented user querying behavior (query profile) and user behavior of accepting results (user profile). To simulate user querying behavior, in each simulation run the query profile was used to generate a set of $k \geq 1$ queries and the queries were submitted to a system in a certain order. To simulate user behavior of accepting results, the model contained a user profile that listed the results that can be accepted and contained the logics that determined whether a result was accepted for the submitted query.

Table 6.1 shows how the user model was applied in the three application scenarios listed in Chapter 5. The first column of the table lists the applications, the second column shows how a query was represented, the third column describes the user profile in the application, and, finally, the fourth column, describes how it was decided whether a result was accepted to the submitted query.

Above, we described general principles of the proposed user model. Let us now consider how the user model is implemented in different applications.

Web search

For the web search application each query consisted of one keyword. The acceptance strategy was modeled as a set of probabilities of choosing a link for a keyword. As links we took the first g links provided by Google for each keyword and the rank of the list was adopted as an identifier. The user profile was built using l keywords q_1, q_2, ..., q_l and determining the probabilities $p(j|q_i)$ of choosing the j-th link, $j \in \{1, \dots, g\}$ while searching with the i-th keyword. We assume that the user accepts one and only one link during the search for the keyword q_i, so $\sum_{j=1}^{g} p(j|q_i) = 1$. The distribution of probabilities in the user model can be seen as a set of association rules with a probability of link acceptance for a given keyword search.

keyword	Google rank of the link									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
tourism	0	0	0.05	0.4	0.05	0.2	0.1	0.05	0.1	0.05
football	0.05	0	0.1	0.3	0.3	0.1	0.1	0.05	0	0
java	0.35	0.3	0.05	0.05	0.05	0.05	0.05	0.1	0	0
oracle	0.1	0.1	0.45	0.2	0	0.05	0.05	0	0	0.05
weather	0	0.3	0	0	0.5	0	0	0.1	0.1	0
cars	0	0	0.05	0.4	0.05	0.2	0.1	0.05	0.1	0.05
dogs	0.05	0	0.1	0.3	0.3	0.1	0.1	0.05	0	0
music	0.35	0.3	0.05	0.05	0.05	0.05	0.05	0.1	0	0
maps	0.1	0.1	0.45	0.2	0	0.05	0.05	0	0	0.05
games	0	0.3	0	0	0.5	0	0	0.1	0.1	0

Table 6.2: A user model for the Implicit system. This model specifies the query profile in the leftmost column and the user profile. Links were different for each keyword and therefore are numbered 1..10.

In the experiments, the number of keywords l was equal to 10, the number of the links provided by Google g was equal to 10. The user profile that was used in the experiment is presented in Table 6.2. The profile has been produced by subjective evaluation of the Google results for five keywords and then duplicating these five rows. The model we used to evaluate the Implicit system is similar to the one proposed in [6].

Software pattern selection

For the IC-Patterns system, we simulated a small community of five developers. Each query in the user model consisted of $l \geq 1$ keywords. A query profile contained a distribution of keywords in user queries, as shown in Table 6.3. A user profile contained m patterns that the user accepted for the queries. Note that in this application the acceptance behavior did not depend on the submitted query. The intuition behind this was as follows: the user has a single problem to solve using patterns, the problem can be described in a number of ways represented by queries generated using the query profile, and the problem can be solved with the use of one of the patterns contained in the user profile.

In the experiments we set the number of patterns $m = 2$ and $l = 3$, so user queries consisted of three keywords that were chosen according to probabilities specified in query profiles. This query length has been chosen because it is very close to the average query length of 2.99 words, reported in one of the recent studies of user behavior in web search [55]. The user profiles for five developers are listed in Table 6.4. For each user there is a profile pattern and several relevant patterns. A *profile pattern* is the pattern used to generate distribution of keywords in queries of a user. *Relevant patterns* are the patterns the user accepted for their queries.

Web service discovery

In the web service discovery application each user query contained a short natural language description of the service functionality. The queries were grouped in five categories and for each category the user profile contained the list of the operations of the web

u01									
access 0.07	processes 0.04	requests 0.06	resources 0.07	users 0.04	you 0.05	monitor 0.04	reference 0.04	authorization 0.05	can 0.04
enforced 0.04	policies 0.05	book 0.04	eduardo 0.04	fernandez 0.05	pattern 0.11	patterns 0.04	security 0.09	titled 0.04	
u02									
connection 0.08	established 0.04	improved 0.04	packets 0.09	security 0.1	system 0.06	can 0.04	correlate 0.06	b 0.04	eduardo 0.04
fernandez 0.04	language 0.04	n 0.06	pattern 0.11	patterns 0.04	petrie 0.04	seliya 0.04	titled 0.04		
u03									
areas 0.05	controlled 0.03	memory 0.03	processes 0.07	should 0.03	system 0.03	users 0.03	access 0.13	address 0.08	control 0.04
space 0.06	virtual 0.05	could 0.03	eduardo 0.03	fernandez 0.04	pattern 0.06	patterns 0.03	security 0.05	systems 0.03	titled 0.04
application 0.03	rights 0.03								
u04									
access 0.06	control 0.06	large 0.03	number 0.06	resources 0.06	system 0.06	users 0.05	you 0.04	based 0.04	many 0.04
objects 0.04	rights 0.04	subjects 0.04	eduardo 0.03	fernandez 0.04	pattern 0.04	patterns 0.04	security 0.08	titled 0.03	group 0.04
roles 0.08									
u05									
access 0.12	control 0.05	created 0.05	objects 0.11	object 0.08	applications 0.04	defined 0.03	must 0.04	rights 0.1	when 0.04
eduardo 0.03	fernandez 0.03	pattern 0.04	patterns 0.03	security 0.05	systems 0.04	titled 0.03	creation 0.04	new 0.05	

Table 6.3: Query profiles for the IC-Patterns system. The query profiles contain the distribution of keywords in user queries. For each user, the query profile contains keywords and probabilities of appearance of the keywords in user queries.

ID	profile pattern	relevant patterns
u01	Reference Monitor	Authorization
		Execution Domain
u02	Stateful Firewall	Packet Filter Firewall
		Proxy Based Firewall
u03	Virtual Address Space Access Control	Controlled Process Creator
		Execution Domain
u04	Role Based Access Control	Authorization
		Multilevel Security
u05	Controlled Object Creator	Controlled Process Creator
		Execution Domain

Table 6.4: The user profiles for the IC-Patterns system.

Category	Web service	Operation
Currency	{http://www.webserviceX.NET/}CurrencyConvertor	conversionRate
	{http://www.xmethods.net/sd/}CurrencyExchangeService	getRate
	{http://www.myasptools.com/}currencyWS	getRate
	{http://www.xignite.com/services/}XigniteCurrencies	getLatestCrossRate

Table 6.5: A user profile for the web service discovery system.

services that provide the required functionality. These operations were accepted when a query from the corresponding category was submitted. An example of user profile for the category "Currency" is given in Table 6.5. The request-generation behavior of the user was simulated by choosing and submitting one of the requests randomly. The result-selection behavior of the user was simulated by choosing and invoking one of the service operations to perform the task. The intuition behind the user profile was as follows: the user submits a request for a service operation. After getting suggestions, the user invokes one of the operations considered relevant to the request.

6.1.3 Dimensions and metrics

In this section, we present the metrics we used to evaluate the quality of recommendations. We also describe which dimensions we used to test the scalability of our recommendation approach.

Since we recommended different things, namely links, patterns, operations of web services, in the three application scenarios, in the following we use the term *item* to refer to the thing which is being recommended. We used the following metrics [10] in order to evaluate the quality of suggestions:

- We call an item **relevant** to a query if it is in a user profile [for the query].

- **Precision** is the ratio of the number of suggested relevant items to the total number of suggested items, relevant and irrelevant.

- **Recall** is the ratio of the number of proposed relevant items to the total number of relevant items.

- **F-measure** is a trade-off between precision and recall. It is calculated as follows:

$$\text{F-measure} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}.$$

The relevance for the Implicit system is defined differently: we call a link **relevant** to a particular keyword if the probability of its acceptance, as specified in the user profile, is greater than some pre-defined relevance threshold t.

We measured the **response time** of a recommendation method as the time passed from the moment the method received the submitted user query till the moment the results were sent to the user. We measured the performance of the IC-Patterns system for each retrieval method, namely for IR, CBR retrieval and the IC-Service recommendations. We also measured the response time of the personal agent as the time passed from the moment the agent received the query till the moment the agent sent results to the user. The response time of the personal agent obviously depends on the load of the system,

but does not depend on the network load. Therefore, this time is shorter than the time between the moment when the user actually submits the query and the moment when the user receives results accessing the system in distributed settings.

We replicated each simulation r times and averaged the values of the performance and quality measures to control the effect of the order and content of queries.

To test the scalability of our approach we used the dimension of the number of agents, i.e. we increased the number of agents in the IC-Patterns system to see how the behavior of the system changes.

6.2   Quality and performance evaluation

In this section, we present the experiments on the evaluation of the performance of the system and the quality of recommendations.

6.2.1   Web search

The goal of the experiment was to understand how the insertion of a new member into a community affects the relevance, in terms of precision and recall, of the links that were produced by SICS.

The interaction between agents and users was replaced with the interaction between agents and user models described in subsection 6.1.2. We set the relevance threshold $t = 0.1$, so a link was considered relevant to a keyword if the probability of accepting the link for the keyword was greater than or equal to 0.1. Since links provided by Google for a certain keyword change their ranks continuously, before the experiment we stored the links corresponding to the chosen keywords in a dataset and replaced querying Google with getting links from this dataset. We did not use the following features of the Implicit system: past search history, the inductive module.

We computed recall in a way which slightly differs from what described in subsection 6.1.3. The total number of relevant links was adjusted by adding links coming from agents and the SICS. Even though such links were already recommended by Google, we counted such repetitions. We did this because in such a way the model of interactions became more similar to a real-life situation, where users (and their agents as well) would have different collections of links, and, therefore the links coming from agents would not be repetitions. However, because of this interpretation of recall, the quality of suggestions was underestimated.

Assuming that all users were members of the same community and had similar interests, the probabilities of accepting links for each user were obtained from the distribution given in Table 6.2 by adding noise uniformly distributed in [0.00,...,0.05]. We added noise to each entry of the table and then renormalized all entries in order to keep the sum of each row equal to one. Using this procedure we generated five different user profiles.

From our set of 10 keywords for each agent we generated 25 sequences of 25 keywords by extraction with repetition. Each sequence was used for modeling the user query behavior in a simulation run. The user acceptance behavior was modeled in the following way. Given a keyword in the sequence of keywords, the accepted result was generated randomly
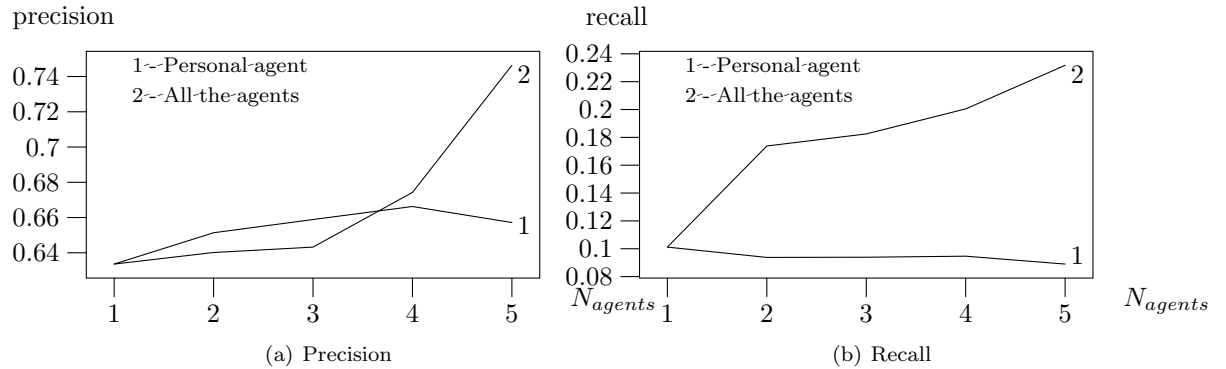
Figure 6.1: Average precision and recall of 25 simulations with different number of agents.

according to the distribution that was specified in the user model. The links from other agents that were equal to the accepted result were marked as accepted, while other links received from the agents were marked as rejected.

In each simulation run we performed 25 searches for each agent in the platform. At the end of each run the observation data were deleted to allow the SICS to produce recommendations from scratch in the next run. We replicated the simulation 25 times in order to control the effect of the order of the keywords and link acceptance. We ran experiments for 1, 2, 3, 4, 5 agents. With one agent on the platform, the agent acted alone without interactions with the others. With five agents we had a small community where agents interacted with each other.

We computed precision and recall of the links proposed by the agents. Line 1 in Figure 6.1(a) represents precision of the links that were recommended by the user personal agent alone. The SICS, which was incorporated in the agent, produced these links by analyzing stored observations. Line 2 represents precision of the links proposed by all the agents including the personal one. The agents were discovered during the external search or were provided by the DF. In Figure 6.1(b) we present analogous curves for recall.

From these figures we can note that the increase in community members caused the increase in the recall of the suggestions from the agents. It is probably due to the fact that when there were more agents in the system, there were also more interactions between them. The agents provided each other only with one link, thus, when more links were provided by the agents during the search, it lead to an increase of the percentage of relevant links proposed by the agents and, therefore, the increase of recall. Moreover, the increase of recall appeared without the decrease of precision which remained at a rather high level ranging from 0.63 to 0.75. The value of recall was also rather good and changed from 0.09 to 0.23. We also studied the statistical significance of the difference between agents with the same distribution of probabilities in different simulations. We performed t-Tests with Bonferroni correction, namely dividing p-value by the number of tests we have performed, in order to control the type I error. These tests proved that the average recall for four and five agents was consistently better ($p < 0.01$) than the average recall of the simulations with smaller number of agents. The results also proved the hypothesis that after a certain number of interactions, agents were able to propose links based on the past user actions.

In other words, the obtained results proved that our way of complementing the Google search engine with suggestions that are produced as a result of indirect user collaboration allows for improving the quality of the web search. An important point is that even without extending query with additional keywords and re-ranking links by the system, it is possible to discover which of the Google links the community prefer and to achieve better quality of suggestions with respect to Google.

6.2.2 Software pattern selection

The overall goal of the experiments was to compare the performance of the system and of the different pattern retrieval methods: IR, CBR, and SICS recommendations. In particular, we would like to measure the impact of the SICS on the system by comparing the performance of the system with and without the SICS.

In the experiments we simulated user behavior with the user model described in subsection 6.1.2. During the experiments, the multi-agent platform contained a special agent, the simulation manager, which was responsible for carrying out the simulation. The task of the simulation manager was to create a number of personal agents, to collect the information regarding searches and to replicate simulation several times. The number of agents and the number of repetitions was specified in the simulation scenario. Personal agents sent the information about the simulated searches to the simulation manager using the FIPA-Request protocol[1].

In the experiments, we did not use the inductive module of the SICS to update the theory and the recommendations were generated entirely by the composer module. Also, models of the users did not produce *reject* actions, just *request* and *apply*.

To build a small community of five developers, we took away five patterns from the repository of security patterns[2] and assigned them to each of the developers as shown in the first two columns of Table 6.4. These patterns were used in order to create query profiles in user models. Each query profile was created as follows: given a document, we computed the frequency of the terms in this document and then saved each term and its normalized frequency as the probability of term's occurrence in the document (see Table 6.3 in subsection 6.1.2. Each user query was generated as a sample from the distribution of terms saved in query profile, thus, a query contained a problem description in free-text form.

To determine the relevant patterns, i.e. those marked as "solution to the problem", corresponding to the profile patterns, the following approach was adopted. We represented each pattern in the repository as *'a bag of words'* [10], i.e. as a vector that contains 1 at the i-th component if the term t_i is present in the pattern description, and 0 otherwise. Then we calculated the similarity between the profile pattern x and the rest of the repository. The cosine similarity metric [10] was used:

$$cos(x, x') = \frac{(x * x')}{(||x|| * ||x'||)}, \tag{6.1}$$

where $||x||$ is the Euclidean norm of the vector representation of the document x.The d

---

[1]FIPA. Foundation for Intelligent Physical Agents. `http://www.fipa.org/`
[2]The repository of security patterns contains 59 patterns.

documents which were the most similar to the profile pattern x, excluding the documents used to create profiles, were put in the profile as relevant patterns. We set $d = 2$ and computed ten relevant patterns corresponding to the profile patterns as shown in the third column of Table 6.4 in subsection 6.1.2. Our approach of taking several patterns from the repository and using them for evaluation was somehow similar to the leave-one-out methodology, where one sample is retained for validation and the others are used for training, therefore we called this experiment leave-profile-patterns-out.

We ran simulations with different number of searches, namely $k = 1, \ldots, 30$, measuring the precision, recall, and F-measure of the recommendations after completing the k searches for every user in each simulation run. We also measured the response time of the system for each query. At the end of each simulation run, the database of observations was deleted in order to have the SICS producing recommendations from scratch. We replicated the simulation 25 times and averaged the precision, recall, F-measure, and the response time to control the effect of the order and keywords of queries.

Since IR and CBR methods were independent and did not use any information neither from each other not from the SICS, from this experiment we could also see what happens in case the retrieval performed by using only IR or only CBR methods. User queries were treated by the retrieval methods differently: for the IR method each keyword was searched in the whole pattern, while for the CBR method each keyword was searched only in the Pattern.Name, Pattern.Context, Pattern.Problem sections of the pattern. For the recommendations produced by the SICS, the similarity between a user query and queries submitted by other users in the past was calculated as the fraction of common keywords. In the experiment we set the threshold of the similarity between user actions in the SICS in such a way that actions were similar if their names were the same, and the queries in actions had at least two keywords in common. In other words, the *request* (*apply*) actions were similar if the problem descriptions in these actions had a non-trivial overlap. This resulted in the similarity threshold $s = 0.7$. Please, note, that focusing on the community as a whole, we did not differentiate between developers when calculating similarity between actions.

The results contain the boxplots for the precision, recall and F-measure of the patterns retrieved using IR and CBR methods, recommended by the SICS, and by the system (together by IR, CBR results and recommendations). They also include the response time of the system, as shown in Figure 6.5. Figure 6.2 shows the precision of the recommendations produced by the five personal agents for five developers. Analogous results were obtained for the recall (Figure 6.3) and F-measure (Figure 6.4).

The figures for precision suggest that the precision of all retrieval methods does not depend much on the number of searches. The precision of the CBR method is the lowest, while the precision of the SICS method is the highest. An interesting observation from these figures is that the precision of SICS stabilizes as the number of searches increases. Consequently, the precision of the whole system stabilizes too.

From the results for recall, it is possible to see that the recall of the IR and the CBR methods remains at the same level, independently of the number of searches. This is reasonable, because these methods do not use any user feedback for improving suggestions. Here we can draw an analogy with conventional search engines whose set of results to the
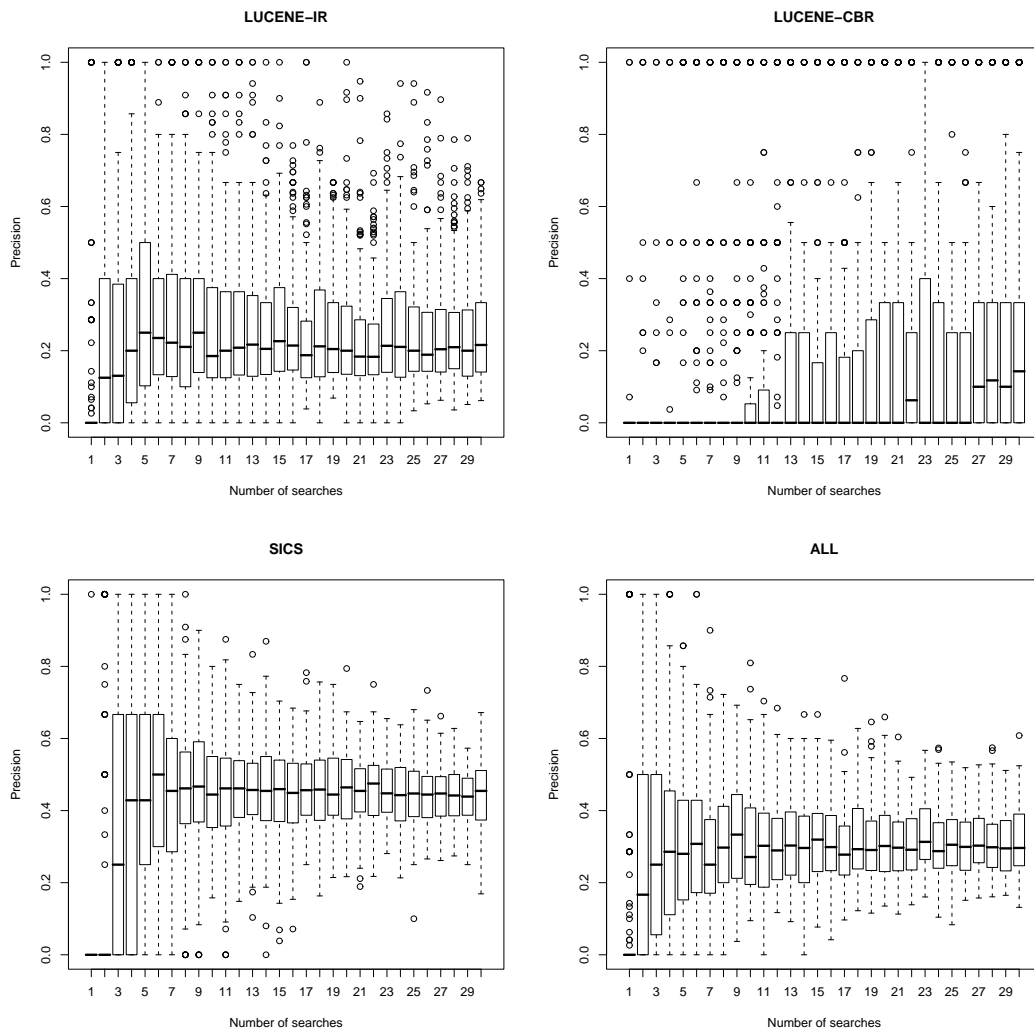
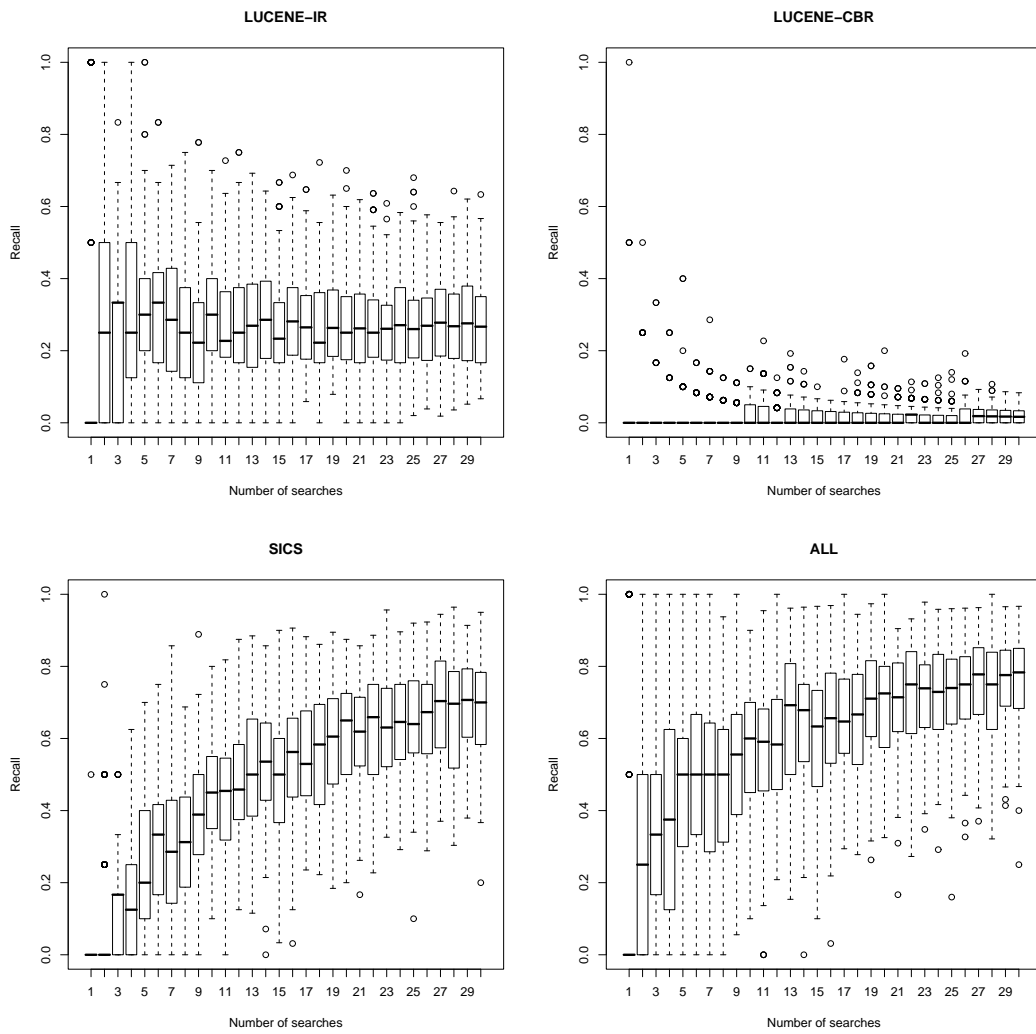Figure 6.2: Precision of the retrieval methods in the IC-Patterns system.

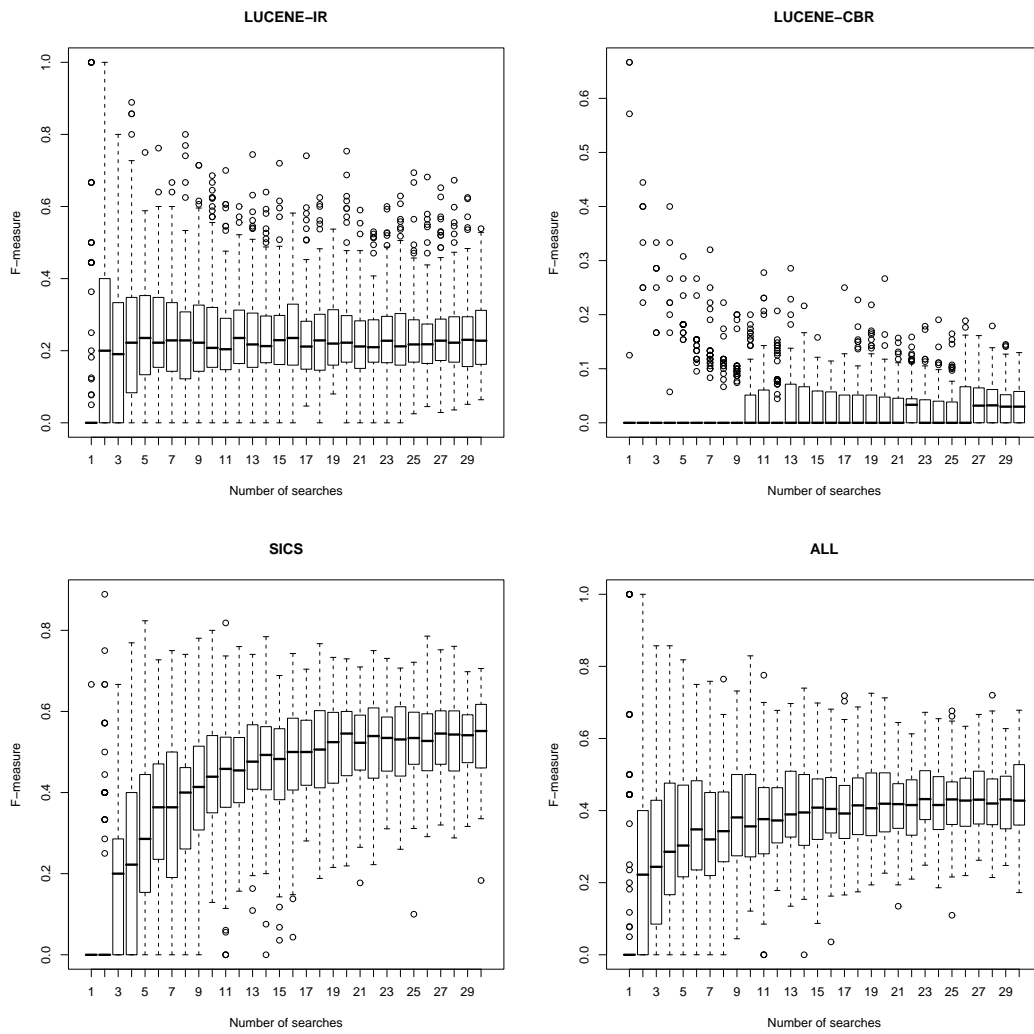Figure 6.3: Recall of the retrieval methods in the IC-Patterns system.

Figure 6.4: F-measure of the retrieval methods in the IC-Patterns system.

same query is generally the same for all users. However, as we can observe in the case of SICS, and, consequently, system recommendations, the use of personalized approaches provides a steady increase in recall. We can observe much lower increase in F-measure of recommendations by the SICS and the system, because it is influenced by precision.

The poor performance of the CBR method is probably explained by the fact that while whole patterns were created for user queries, only portions of patterns were indexed by the CBR method and used for the retrieval.

Figure 6.5 shows the response time of the retrieval methods in milliseconds. As we can see, the response time of the IR and CBR methods is lower that the response time of the SICS. Moreover, those response time do not depend much on the number of searches, while the response time of the SICS does depend a little. This is probably explained by the number of observations growing when simulation run contained more searches. The outliers for each point in SICS response time correspond to the initialization of the SICS, which takes some time. However, as shown by means, once initialized, the SICS responds faster. Similar behavior is observed for the IR algorithm: when there is only one search, the algorithm performs much slower because it initializes the index in each simulation run.

Figures contain a lot of outliers that correspond to the cases of initializing recommendation methods in different runs for different agents, or occur in case of too short runs, e.g. consisting of only several searches.

6.2.3 Web service discovery

In the experiment we used a collection of 20 web services from the xMethods service registry[3] divided into five topic categories. For each category we found four semantically equivalent operations and formed 20 requests based on their short natural language descriptions from WSDL files. The interaction of users with the system was simulated using the user model defined in subsection 6.1.2. The invocations of web services by user models were monitored by the SICS Remote Client. We used two metrics for calculating similarity between user requests: a syntactic metric, the Vector Space Model (VSM) with Term Frequency - Inversed Document Frequency (TF-IDF), and a semantic metric, Word-Net-based.

We ran simulations with the number of searches $k = 100$, and the number of users equal to 4 and to 20, measuring the precision, recall, and F-measure of the recommendations in each simulation run. The results are shown in Figure 6.6. 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 fault 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.
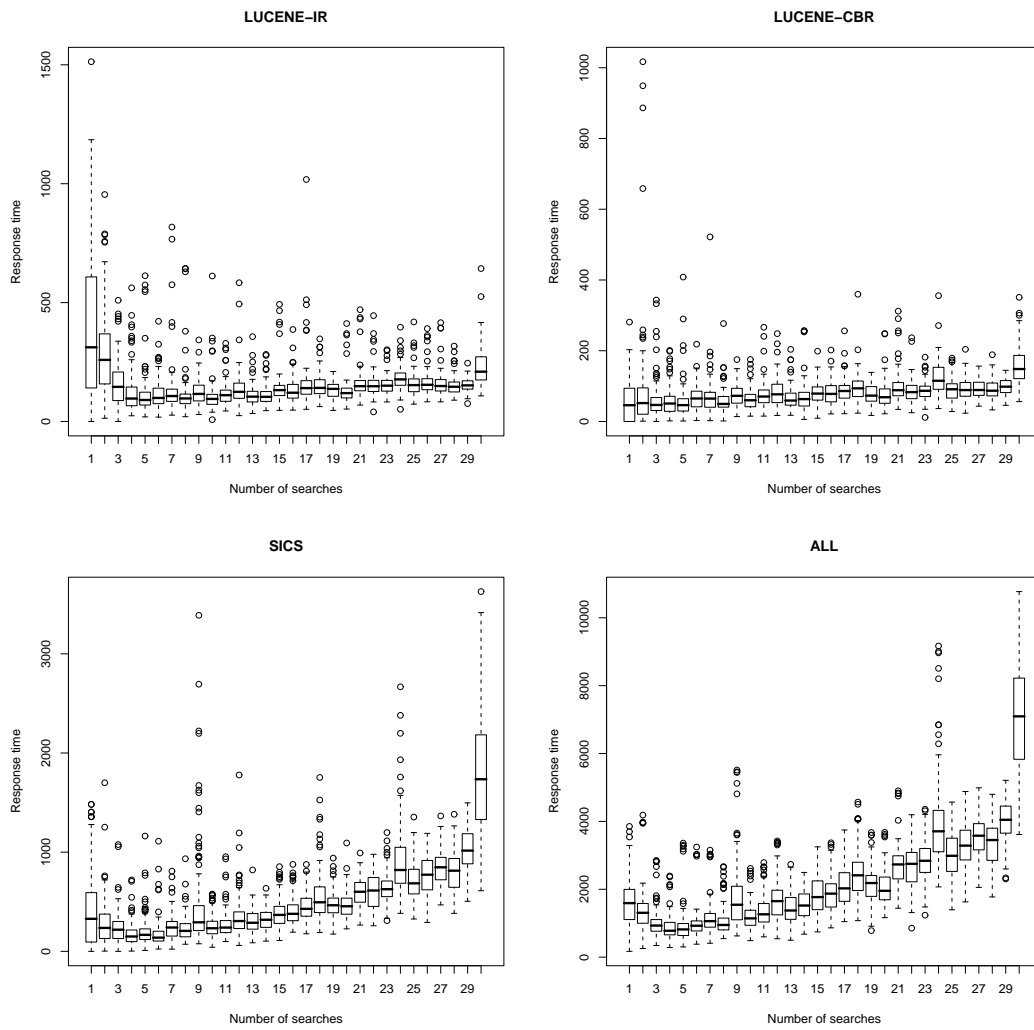
---

[3]http://xMethods.com/

Figure 6.5: Response time of the retrieval methods in the IC-Patterns system.

(a) VSM with TF-IDF, 4 clients

(b) WordNet-based semantic similarity metric, 4 clients

(c) VSM with TF-IDF, 20 clients

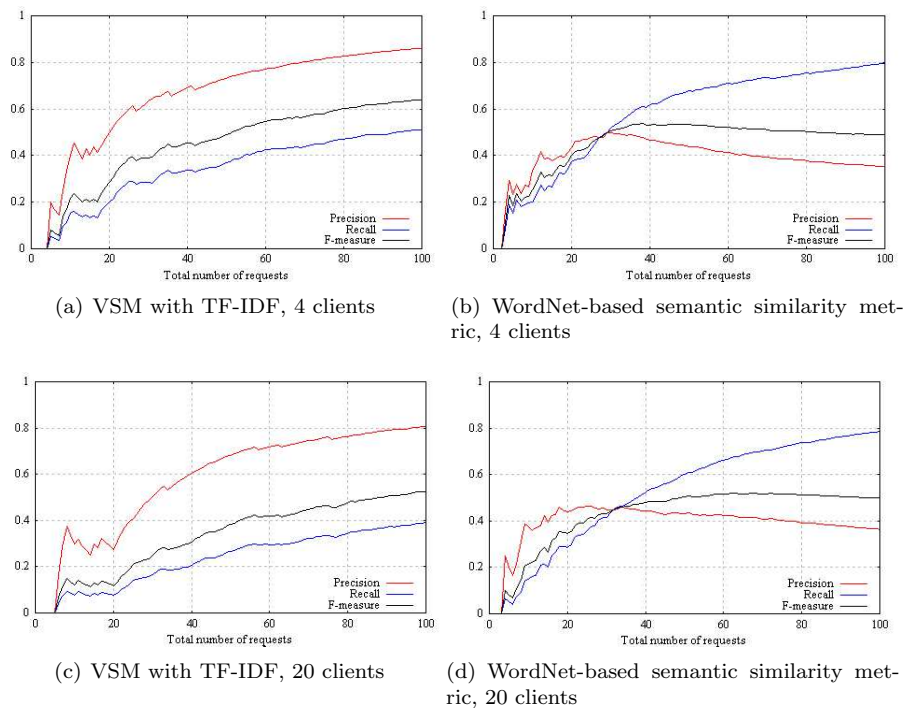(d) WordNet-based semantic similarity metric, 20 clients

Figure 6.6: Performance of the web service discovery system.

6.3 Evaluation with real users

In this section, we report the results of the evaluation of the Implicit system with real users. The goal of the evaluation with real users was to see how the system is accepted by people and whether it provides enough support for sharing experience in a community of practice.

We performed the evaluation in a small company located near Trento, Italy. The company offers courses in different areas of IT. For promoting interaction between course attendees the company provided a forum with topics dedicated to the content of the courses.

For the described evaluation people attending the courses related to the Java programming language were selected. Eleven course attendees agreed to participate in the study. When dealing with the system, the participants were focused on a narrower topic of game development in the Java language. The Implicit system was integrated in the company forum, namely, the interface was modified to enable user requests, display suggestions from the Implicit system, and permit providing feedback submission to the posts.

Participants used the system in the following way: whenever they wanted to get an answer to a question, they clicked on the Implicit button in the interface and submitted several keywords as their query to the system. The system ran the general search mechanism described above, forwarding the query to other personal agents and to Google. In addition to this, the query was forwarded to the specific wrapper agent that dealt with the retrieval of the forum messages. This wrapper used the SICS to retrieve messages that were accepted for similar queries previously. The feedback collected previously has been

Period	number of requests
Weeks 1 and 2	10
Weeks 3 and 4	18
Weeks 5 and 6	60

Table 6.6: The number of requests to the system.

Number of accepted results	number of requests
1	32
2	16
3	2
4	2

Table 6.7: The number of accepted results.

used to rank the results. In this application, first forum posts, then links from the users, and, finally, links from Google were displayed and a result was considered "accepted" if the user clicked on it. Moreover, users were able to express their level of satisfaction with results by means of explicit feedback — for each post the feedback contained ratings in the scale from 1(very bad) to 6(excellent) for the five following categories: quality of message, quality of the code, was message was useful to the user, would the user recommend the message to others, was it relevant to the original question of the user. The users were not required to provide ratings in all five categories. There was also a possibility to rate messages directly when browsing forum threads. Messages rated in such a way were retrieved by the forum wrapper in case the retrieval of messages using the SICS produced no results. This functionality helped us to deal with the cold-start problem [28].

We ran the system for six weeks and measured the number of queries submitted and the number of recommendations accepted. The results containing the number of queries submitted during the evaluation are given in Table 6.6. As we can see, the users were reluctant to use the system during the first two weeks, while during the second two weeks the number of queries slightly increased and reached the number of 60 queries in the final period.

Table 6.7 show the how many times one, two, three, and four results were accepted. As can be seen from the table, in most cases users accepted only one result, sometimes two, and only rarely three or four. For the total of 88 queries, in 52 searches users accepted at least one result from the system, i.e. in 60 per cent of the searches the system provided something of interest.

The obtained results show that the system, after some period, was accepted by the community and was able to provide useful recommendations.

6.4   Scalability evaluation

We performed the study on the scalability of the IC-Patterns system.

The goal of this experiment was to test how the system response time changes depending on the number of users. Although we have recorded the performance of the IR

and the CBR algorithms, no significant dependency on the number of users should be observed. However, since there is only one SICS for answering all user queries, we expect the response time to decrease as the number of users increases.

All materials of the experiment were the same as in the experiment in subsection 6.2.2.

In this experiment we copied the five user models from subsection 6.2.2 several times to simulate the community of u users. Of course, this lead to appearance of several chunks of very similar users, but in this experiment the goal was not to focus on the reality of such a community, but merely on the size of the community.

In the experiment we set the number of searches $k = 15$ and we ran simulations with the number of users, u, equal to 3, 6, 9, 12, 15, 18, measuring the response time of the system for each query. We replicated the simulation 25 times and averaged the response time.

The results contain the response time of the system as shown in Figure 6.7. As can be expected, the response time grows with the number of agents. The response time of the IR and CBR methods changes linearly, while the response time of the SICS more resembles quadratic dependence. However, up to nine agents, the response time of SICS is less than or equal to a couple of seconds. Let us consider that nine agents in this experiment do not really correspond to nine users, but, rather, they represent nine users *continuously* and *simultaneously* querying the SICS. This can hardly be considered as a realistic scenario. Thus, we can claim that the system can support more users as long as no more than nine users query our system simultaneously. The number of user that can be served is subject of further investigation. However, this defines a kind of throughput threshold for the system when used with xml files for storing observations. As we described in Section 4.5, the use of a relational database in the system would allow for processing queries faster, and should permit serving more users simultaneously.

6.5 Discussion

In this chapter, we have presented the objectives and the methodology of the evaluation of the applications described in Chapter 5. We have developed a user model, because there is no commonly accepted user model for benchmarking recommendation systems in different domains. However, the model we used in Implicit is similar to the one proposed in [6].

In the following, we discuss the results of the evaluation in the context of the objectives.

We have tested the performance of the IC-Patterns system, measuring the response time of different recommendation methods. The response time of the SICS was lower than the response time of the IR and CBR methods, but the SICS showed better quality of recommendations. Overall response time of the system depended on the history of observations and the number of agents, and in most cases ranged from 0.9 to 3 seconds, which is an acceptable result.

We have tested the quality of recommendations of the systems, and compared the retrieval performance of different algorithms. The results have shown that our systems clearly learns from past interactions. Precision mainly varied from 0.3 till 0.8, while recall varied from 0.18 till 0.7, depending on the scenario. Overall, the results improved over
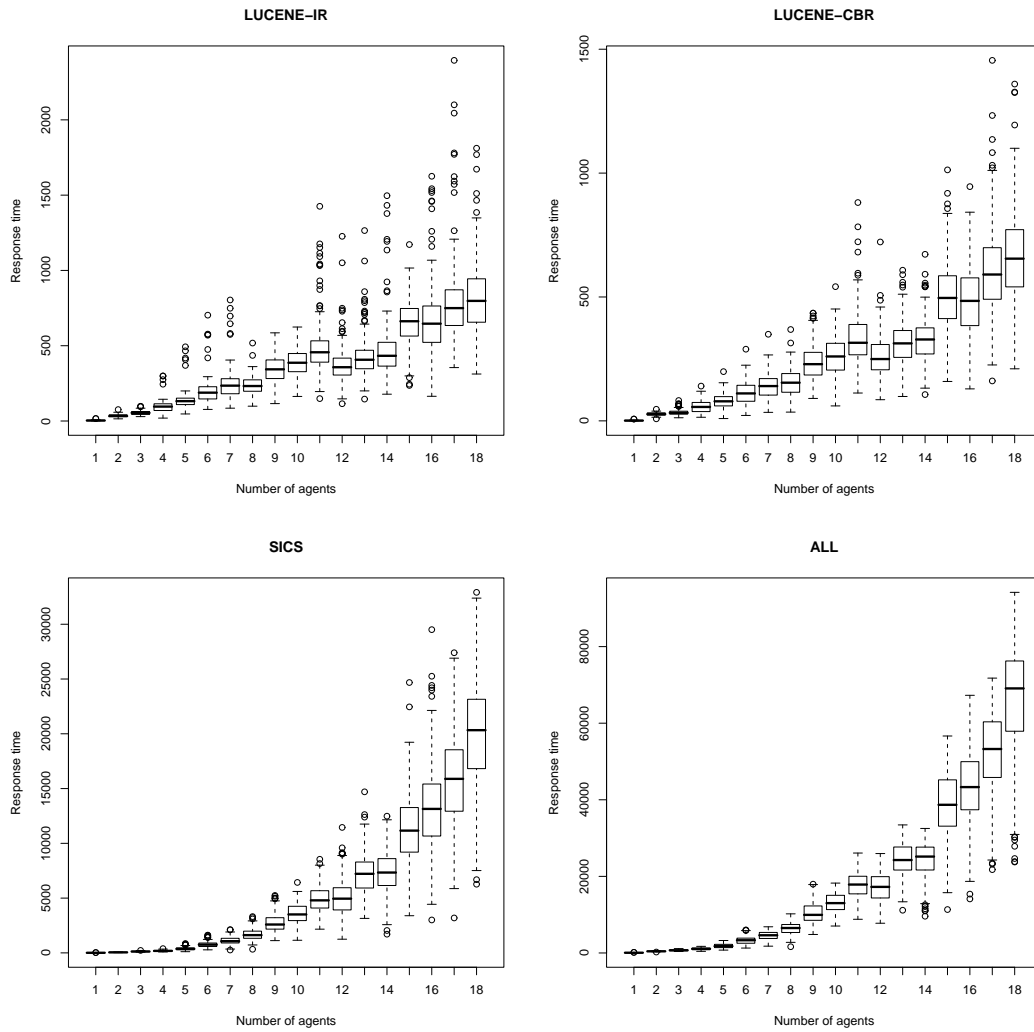
Figure 6.7: Dependence of the response time of the retrieval methods on the number of agents in the IC-Patterns system.

the first 10 to 20 searches and then stabilized at the reached high level.

We have measured the impact of the SICS in each system by comparing the quality of suggestions with and without the SICS. The results suggested that the effect of improving the quality of the recommendations over time was caused by the SICS. In most cases, the SICS outperformed conventional retrieval methods in terms of precision, reaching the level of 0.55-0.85, depending on the application. Moreover, while the recall of the conventional retrieval methods remained more or less at the same level, the recall of the SICS recommendations increased. However, the SICS recommendations are based on the actions users performed on the recommendations produced by other methods, and, in a way, are not possible without them. Therefore, conventional methods (Google links in Implicit, IR and CBR methods in IC-Patterns, random choice in the web service discovery system) are required as a kind of bootstrapping for the systems.

We have tested the scalability of the IC-Patterns system. Since we used XML files for storing data about past user actions and did not perform optimization, the response time of the SICS was acceptable only in small communities (up to nine agents). However, we should consider that nine agents in the experiment did not really correspond to nine users, but, rather, they represented nine users *continuously* and *simultaneously* querying the SICS. This can hardly be considered as a realistic scenario, and we expect that even in the present configuration the system will be able to handle more users. Moreover, we expect that the use of a relational database in the IC-Service will improve the scalability of the approach and greatly improve the performance of the system.

Finally, the results of the evaluation of the Implicit system with real users have shown that even though users were initially reluctant to using the system, the number of queries slightly increased later and finally reached the number of 60 queries, which is almost six queries per person. Moreover, in 60 per cent of searches the system provided something of interest, because users accepted at least one result from the system. Thus, we can conclude that the Implicit system attracted the critical mass of the community members and was perceived useful.

Chapter 7

Related work

In this chapter, we review the related work. In Section 7.1 we discuss the applicability of the Implicit Culture Framework in the field of recommendation systems. In Section 7.2 we see how our approach is related to collaborative filtering. In Section 7.3 and Section 7.4, respectively, we discuss similarities with stigmergy and social navigation approaches.

7.1   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" [28]. The baseline 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 [101], stereotypes [92], and implicit or explicit feedback from users [55]. 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.

The main approaches for the development of recommendation systems are: content-based and collaborative filtering [113]. In the content-based approach, the content of the items is analyzed, while in the collaborative filtering the ratings of different users are compared to produce recommendations. There are also hybrid approaches that combine both [9, 28]. Most of the publicly available recommendation engines (e.g. Taste[1], CoFE[2], Alkindi[3]) implement collaborative filtering techniques. There are also tools that provide content-based recommendations (Recommender.org[4]), and domain-specific recommendation tools (The Scout Portal Toolkit[5]).

Many of the existing recommendation systems use explicit feedback [84, 126, 141], while people are not usually willing to actively provide feedback [37, 120]. Furthermore, sometimes users are inconsistent in the explicit ratings provided [82]. All these facts

---

[1]`http://taste.sourceforge.net/`
[2]`http://eecs.oregonstate.edu/iis/CoFE/`
[3]`http://mappa.mundi.net/signals/memes/alkindi.shtml`
[4]`http://sourceforge.net/projects/recommender-org/`
[5]`http://scout.wisc.edu/Projects/SPT/`

item	Kevin	Lucy	Max	Naoko
1	1	4	2	2
2	5	2	4	4
3	?		3	
4	2	5		5
5	4	1		1
6	?	2	5	

Table 7.1: An example of matrix of ratings.

suggest that implicit indicators of user interests should be exploited. Moreover, the study by Fox et al. [55] has shown that implicit measures can be a suitable alternative to explicit feedback.

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. To some extent, the applications presented in Chapter 5 adopt this approach.

7.2   Collaborative filtering

Collaborative filtering [113] 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. For instance, in Table 7.1, Kevin rated items 1, 2, 4, 5, and has not seen items 3 and 6. In order to see which of the unseen items he will like more, the collaborative filtering algorithm calculates the prediction of Kevin's ratings on items 3 and 6, using weighted similarity between Kevin and other users. This similarity depends on the correlation between the ratings of Kevin and other users.

Among the most influential papers on collaborative filtering we should mention those of GroupLens group[6], including systems for recommending news (GroupLens) [84], movies (MovieLens[7]), research papers (TechLens) [141]. One of the probably best-known real-world applications of collaborative filtering is Amazon.com [90], where it is used for recommending products.

The Implicit Culture Framework is claimed to be more general than collaborative filtering [21] since it filters not only ratings, but actions in general, with *rate* being only a particular example 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

---

[6]GroupLens Research. `http://www.grouplens.org/`
[7]Movie recommendations. `http://www.movielens.org/`

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.

7.3 Stigmergy

Stigmergy was introduced in biology as an approach for explaining self-organization, i.e. emergence of global behavior patterns from local interactions [29]. More specifically, it explains social mechanisms of coordination based on interaction through local modifications to a shared environment. Examples of such coordination include the movement of wooden chips by termites and ant foraging, i.e. the deposition of pheromone by ants so that other ants can follow them.

Stigmergy has been widely applied in multi-agent systems for implementing emergent coordination. However, as pointed out by Ricci et al. [114], existing approaches mostly consider primitive societies composed of a large amount of simple, ant-like, non-rational agents. Therefore, the authors propose a notion of cognitive stigmergy, which exploits the notion of complex cognitive artifacts populating agent environment and used for coordination. The notion of artifact is very important for further development of the Implicit Culture Framework. On the one hand, while now we consider transmission as given, it can be explained and clarified by using the notion of artifacts as facilitators or enablers of the transmission. On the other hand, the notion of object, which is used in the detailed SICS architecture, can be seen as a particular case of artifacts. Thus, the analysis of actions taken by agents on a set of objects, performed by the inductive module, can be seen as an attempt for understanding how artifacts influence agents' actions.

In the field of recommendation and personalization systems, the stigmergy approach is often called swarm intelligence. Several swarm intelligence theories, e.g. trail laying, and ant foraging, have been applied for guiding people (e.g. learners) to relevant information using the data from previous interactions with the system [46, 144]. These approaches suggest that the user follow trails taken by the majority. Differently, in the Implicit Culture approach the user's actions are compared with actions of the whole community and not necessarily the most popular ones are suggested. More precisely, taking into account past actions of the user, the system can offer actions which are less popular in general, but are common among the part of the community which is the most similar to the user.

7.4 Social navigation

Implicit Culture in general and applied in recommendation systems in particular is related to the notion of Social Navigation. Initially introduced in [44] and further developed by Dieberger et al. [41] it has been also applied to the problem of the navigation on the Internet [40].

The main objective of Social Navigation is to help people to take decisions by using, directly or indirectly, information from other people. With this in mind, Dieberger et al. [41] introduce several styles of Social Navigation systems: *recommendation systems*, which help people to make a choice by looking at what other people with similar interests

have chosen; *populated spaces*, which use the idea of a populated space in which other people can be encountered; and *"history-enriched" systems*, which use the history of previous actions to guide the user. Thus, the problem targeted by Social Navigation systems can be formulated as guiding people to relevant information. This problem is similar to the one addressed by applications of the Implicit Culture in recommendation systems, as described in Chapter 5. However, the general idea of the Implicit Culture approach, i.e. transferring culture of a community, is much broader. The same applies to the problem addressed by this thesis, that is, a narrower problem of the transfer of behavior from one group to another. In this perspective, the scope of the Implicit Culture approach seems to be more general than just dealing with the transfer of information between groups. However, when applied in the field of recommendation systems, two approaches look very similar.

On the one hand, Implicit Culture does not cover all the scope of the Social Navigation problems, and it has been defined as a relation between groups of agents. The definition emphasizes the transfer of behavior between groups and allows one to evaluate whether the implicit culture relation arises from some system usage. It can be argued that some effective Social Navigation systems produce such a relation. On the other hand, other Social Navigation systems, for instance, those supporting awareness (e.g. social proxies [48]) do not necessarily produce the implicit culture relation.

On the other hand, establishing the implicit culture relation by means of SICSs can be considered as a particular case of Social Navigation. Implicit Culture-based systems manifest the two key properties of the Social Navigation phenomenon, namely *personalization* and *dynamism* [41]: when producing suggestions, the SICS focuses on a particular person and the situation this person currently encounters; suggestions of the SICS can change as new actions become common for the observed group in the same situations (the evolution of the theory). Also, both Implicit Culture and Social Navigation process user feedback in order to support information navigation. However, differently from Social Navigation, Implicit Culture allows for a formal description of the navigation process.

Finally, we would like to stress that although the SICS encourages the desired behavior of the community members, it does not control the decision-making process and it is the user who takes the final decision. The importance of all this with respect to Social Navigation in the online world has been described in Dieberger et al. [41].

7.5   Concluding remarks

In this chapter, we have briefly reviewed the related work, focusing on recommendation systems, collaborative filtering, stigmergy, and social navigation.

Chapter 8

Conclusion

The main objective of this thesis was to address the problem of discovering, representing, and transferring culture in communities, such as communities of practice or Web 2.0 communities.

We have reviewed and discussed limitations of the existing computer science approaches for dealing with culture, for transferring knowledge, behavior, and culture. We have also studied how the important implicit-explicit dichotomy (or duality, as suggested by some authors) of knowledge and culture is addressed in the literature.

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.

First, we have developed a formalism for defining and representing culture of communities. This included the definition of culture of a set of agents, the classification of problems involving culture, and metrics for measuring, assessing, and characterizing culture. We have shown that the provided formalism is operational, i.e., as we illustrated by a case study, it can be used to compute and measure culture.

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, a general architecture of SICS for behavior transfer, a detailed architecture of SICS modules, 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. In this thesis, we have described all the components of the framework in detail.

We have applied the proposed approach in the domain of recommendation systems to develop three systems: Implicit, a system for recommending web links; IC-Patterns, a system that recommends software patterns; and a system for web service discovery. The systems were implementing different architectures: a multi-agent architecture has been chosen for the Implicit and IC-Patterns systems, and s service-oriented architecture has been used for the system for web service discovery. The applications followed the principle of non-intrusiveness, i.e., integration in the community practices and using implicit

feedback from the users when possible.

We have presented the evaluation methodology and performed evaluation of the developed applications with real users and with simulated user models. The results of the evaluation have shown that the quality (measured as precision and recall) of recommendations of the SICS increased as the number of the community members or the number of searches increased. The precision reached the level of 0.8, and recall reached the level of 0.7, depending on the application. In most cases the SICS outperformed conventional retrieval methods, such as IR or CBR. The response time of the SICS in small communities was acceptable even when XML files were used for storing data about past user actions. We expect that the use of a relational database in the IC-Service will improve the scalability of the approach and greatly improve the performance of the system, as shown by preliminary experiments. The real-user evaluation of the Implicit system has shown that the system attracted the critical mass of the community members and was perceived useful. Overall, we can say that applications have shown how the use of our approach leverages community culture to provide more social benefits (expressed in terms of quality of the recommendations) to communities.

Finally, we have reviewed a number of research fields that are related to our approach: collaborative filtering, recommendation systems, stigmergy, and social navigation.

8.1   Future work

Future work will follow different directions. First, we would like to extend the Implicit Culture Framework to deal with more general problem of behavior transfer, and with the problem of knowledge transfer. Second, we are planning to develop and test inductive module that uses more sophisticated algorithms of the SICS architecture and to apply the framework in other domains, e.g. software testing. For some applications we foresee the need of integration with other approaches. Third, we will develop tools for solving some of the problems involving culture from the proposed classification. In particular, tools for discovering culture of communities, and clustering communities based on their culture can be developed. Finally, we would like to use measures we introduced in Section 3.5 to test the results of the application of our framework. For instance, we can check if a SICS improves homogeneity of a culture and of a group, and to measure the degree and the speed of culture transfer.

8.2   Dissemination of results

The results of the thesis have been published in the IEEE Software journal, in the proceedings of AAMAS'05, AAMAS'06, ISC'06, BIS'07, ACM-SAC'07 conferences and have been presented at the following workshops: Workshop on Multi-Agent Information Retrieval and Recommender Systems at IJCAI'05, Workshop on the Social Navigation and Community-Based Adaptation Technologies at AH'06, Workshop on Wikis for Software Engineering at ACM WikiSym'07. As an extension of the work on the system for recommending software patterns, a focus group on pattern repositories was

organized at the 12th European Conference on Pattern Languages of Programs (Euro-PLoP'07), and a wiki for hosting collections of software patterns (available at `http://www.patternforge.net/wiki/index.php?title=Main_Page`) was developed together with other members of *PLoP community, which includes pattern writers, researchers, and practitioners. Journal papers derived from the results of the thesis are being prepared for submission to the Journal of Autonomous Agents and Multi-Agent Systems, the Journal of Artificial Intelligence, and the Journal of Decision Support Systems. Conference papers have been submitted to UMAP'09 and EuroPLoP'09.

The IC-Service has been applied in several student projects (see the list at `http://disi.unitn.it/~birukou/index.php?page=students`), in particular, for the development of a system that helps to leverage knowledge of a community of biologists [111], a self-organizing system for traffic management [15], and an adaptive system for Internet radio[133]. The IC-Service is being prepared to be released as an open source project.

We are working on the development of an integrated service supporting the whole lifecycle of software patterns The system is a mash-up of several pattern-related services for pattern retrieval, tagging, application and detection in code. It is targeted at and is being developed in close collaboration with *PLoP community.

A pattern for collaborative service discovery[1] has been derived from the web service discovery application, and included to NEXOF-RA project. NEXOF-RA is a part of the Networked European Software and Services Initiative (NESSI), a European research collaboration network. The goal of NEXOF-RA is to build an Open Reference Architecture for the developers and architects of service-oriented systems.

---

[1]Position paper is available at `http://www.nexof-ra.eu/?q=rep/term/158`, the patterns will be released in the nearest future.

Bibliography

[1] Agnar Aamodt and Enric Plaza. Case-based reasoning: Foundational issues,methodological variations, and system approaches. *AI Communications*, 7(1):39–59, March 1994.

[2] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Procedings of the 20th International Conference on Very Large Data Bases (VLDB)*, pages 487–499. Morgan Kaufmann, 1994.

[3] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, and S. Angel. *A pattern language*. Oxford University Press, 1977.

[4] R. Almeida and V. Almeida. A community-aware search engine. In *Proceedings of the 13th international conference on World Wide Web*, 2004.

[5] Yariv Aridor and Danny B. Lange. Agent design patterns: elements of agent application design. In *AGENTS '98: Proceedings of the second international conference on Autonomous agents*, pages 108–115, New York, NY, USA, 1998. ACM.

[6] Kit G. August, Mark H. Hansen, and Elizabeth Shriver. Mobile web searching. *Bell Labs Technical Journal*, 6(2):84–98, 2001.

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

[8] Garrick Bailey and James Peoples. *Introduction to Cultural Anthropology*. Wadsworth Publishing, September 1998.

[9] Marko Balabanović and Yoav Shoham. Fab: content-based, collaborative recommendation. *Commununications of the ACM*, 40(3):66–72, 1997.

[10] Pierre Baldi, Paolo Frasconi, and Padhraic Smyth. *Modeling the Internet and the Web: Probabilistic Methods and Algorithms*. Wiley, 2003.

[11] Wolfgang Balzer and Raimo Tuomela. Collective intentions and the maintenance of social practices. *Autonomous Agents and Multi-Agent Systems*, 6(1):7–33, January 2003.

[12] P Baumard. *Tacit knowledge in organizations*. London & Thousand Oaks: Sage, 1999.

[13] Fabio L. Bellifemine, Giovanni Caire, and Dominic Greenwood. *Developing Multi-Agent Systems with JADE (Wiley Series in Agent Technology)*. Wiley, April 2007.

[14] S. Bender and A. Fish. The transfer of knowledge and the retention of expertise: the continuing need for global assignments. *Journal of Knowledge Management*, pages 125–137, 2000.

[15] Massimiliano Bernabe. Implicit culture for self-organizing systems: a concrete scenario and its performance analysis. available at `http://disi.unitn.it/~birukou/students/theses/MassimilianoBernabe-MS-2007.pdf`. Master thesis, 2007.

[16] Aliaksandr Birukou, Enrico Blanzieri, Vincenzo D'Andrea, Paolo Giorgini, Natallia Kokash, and Alessio Modena. *IC-Service*: A service-oriented approach to the development of recommendation systems. In *Proceedings of the ACM Symposium on Applied Computing (SAC)*, pages 1683–1688. ACM Press, 2007.

[17] Aliaksandr Birukou, Enrico Blanzieri, and Paolo Giorgini. Implicit: An agent-based recommendation system for web search. In *Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 618–624. ACM Press, 2005.

[18] Aliaksandr Birukou, Enrico Blanzieri, and Paolo Giorgini. Choosing the right desing pattern: the implicit culture approach. In *Proceedings of the fourth Industrial Simulation Conference 2006 (ISC-2006).*, pages 55–57. EUROSIS, June 2006.

[19] Aliaksandr Birukou, Enrico Blanzieri Vincenzo D'Andrea, Paolo Giorgini, and Natallia Kokash. Improving web service discovery with usage data. *IEEE Software*, 24(6):47–54, 2007.

[20] Aliaksandr Birukov, Enrico Blanzieri, and Paolo Giorgini. Implicit: A recommender system that uses implicit knowledge to produce suggestions. In *Proceedings of the Workshop on Multi-Agent Information Retrieval and Recommender Systems at IJCAI-05*, 2005.

[21] Enrico Blanzieri and Paolo Giorgini. From collaborative filtering to implicit culture: a general agent-based framework. In *Proceedings of the Workshop on Agents and Recommender Systems*, Barcelona, 2000.

[22] R. Bova, H. Paik, B. Benatallah, L. Zeng, and S. Benbernou. Task memories and task forums: A foundation for sharing service-based personal processes. In *Proceedings of the International Conference on Service Oriented Computing (ICSOC)*, 2007.

[23] Robert Boyd and Peter J. Richerson. *Culture and the Evolutionary Process*. University of Chicago Press, 1988.

[24] Adda B. Bozeman. *Strategic Intelligence and Statecraft: Selected Essays*. Washington, DC: Brassey's, 1992.

[25] Christoph Brumann. Writing for culture: Why a successful concept should not be discarded [and comments and reply]. *Current Anthropology*, 40, 1999.

[26] Peter Brusilovsky, Rosta Farzan, and Jae wook Ahn. Comprehensive personalized information access in an educational digital library. In *JCDL '05: Proceedings of the 5th ACM/IEEE-CS joint conference on Digital libraries*, pages 9–18, New York, NY, USA, 2005. ACM Press.

[27] Tung Bui and Jintae Lee. An agent-based framework for building decision support systems. *Decision Support Systems*, 25(3):225–237, April 1999.

[28] Robin Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370, 2002.

[29] Scott Camazine, Jean-Louis Deneubourg, Nigel R. Franks, James Sneyd, Guy Theraula, and Eric Bonabeau. *Self-Organization in Biological Systems*. Princeton University Press, 2003.

[30] Douglas K. Candland. *Feral Children and Clever Animals: Reflections on Human Nature*. Oxford University Press, USA, October 1995.

[31] Kathleen Carley. A theory of group stability. *American Sociological Review*, 56(3):331–354, 1991.

[32] Kathleen M. Carley and Michael J. Prietula, editors. *Computational Organization Theory*. Routledge, 1994.

[33] Cristiano Castelfranchi. Modelling social action for ai agents. *Artificial Intelligence*, 103(1-2):157–182, August 1998.

[34] Marcelo Cataldo, Kathleen Carley, and Linda Argote. The effect of personnel selection schemes on knowledge transfer. In *In Proceedings of the 9th International Conference on Computational Analysis of Social and Organizational Systems*, 2000. A longer version available as a techreport at `http://www.casos.cs.cmu.edu/publications/papers/marcelo_paper.pdf`.

[35] Michael Chau, Daniel Zeng, Hsinchun Chen, Michael Huang, and David Hendriawan. Design and evaluation of a multi-agent collaborative web mining system. *Decision Support Systems*, 35(1):167–183, 2003.

[36] Liren Chen and Katia Sycara. Webmate: a personal agent for browsing and searching. In *AGENTS '98: Proceedings of the second international conference on Autonomous agents*, pages 132–139, New York, NY, USA, 1998. ACM Press.

[37] Mark Claypool, Phong Le, Makoto Wased, and David Brown. Implicit interest indicators. In *International Conference on Intelligent User Interfaces*, pages 33–40. ACM Press, 2001.

[38] M.D. Cohen, J.G. March, and J.P. Olsen. A garbage can model of organizational choice. *Administrative Sciences Quarterly*, 17(1):1–25, 1972.

[39] Francisco Curbera, Donald F. Ferguson, Martin Nally, and Marcia L. Stockton. Toward a programming model for service-oriented computing. In *ICSOC: Proc. of the 3d Int. Conference on Service-Oriented Computing*, volume 2172 of *LNCS*, pages 33–47. Springer, 2005.

[40] Andreas Dieberger. Supporting social navigation on the world wide web. *Int. J. Human-Computer Studies*, 46:805–825, 1997.

[41] Andreas Dieberger, Paul Dourish, Kristina Hook, Paul Resnick, and Alan Wexelblat. Social navigation: Techniques for building more usable systems. *interactions*, 7(6):36–45, 2000.

[42] Vania Dimitrova, Manolis Tzagarakis, and Julita Vassileva, editors. *SociUM: Adaptation and Personalisation in Social Systems: Groups, Teams, Communities*, 2007.

[43] Tung T. Do, Manuel Kolp, and Alain Pirotte. Social patterns for designing multi-agent systems. In *Proceedings of the Fifteenth International Conference on Software Engineering & Knowledge Engineering (SEKE'2003)*, pages 103–110, 2003.

[44] Paul Dourish and Matthew Chalmers. Running out of space: Models of information navigation, short paper. In *HCI'94*, Glasgow, August 1994.

[45] Hubert L. Dreyfus and Stuart E. Dreyfus. *Mind over machine: the power of human intuition and expertise in the era of the computer.* The Free Press, 2000.

[46] Jon Dron. Control, termites and e-learning. In *Proc. of IADIS International Conference Web Based Communities 2005*, pages 103–110, 2005.

[47] Joshua M. Epstein and Robert L. Axtell. *Growing Artificial Societies: Social Science from the Bottom Up.* The MIT Press, November 1996.

[48] Thomas Erickson and Wendy A. Kellogg. *Designing Information Spaces: The Social Navigation Approach*, chapter Social Translucence: Using Minimalist Visualisations of Social Activity to Support Collective Interaction, pages 17–41. Springer Verlag, 2003.

[49] Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning About Knowledge.* The MIT Press, 1995.

[50] Weiguo Fan, Michael Gordon, and Praveen Pathak. On linear mixture of expert approaches to information retrieval. *Decision Support Systems*, 42(2):975–987, November 2006.

[51] Rosta Farzan and Peter Brusilovsky. Community-based conference navigator. In *Proceedings of SociUM Workshop (1st Workshop on "Adaptation and Personalisation in Social Systems: Groups, Teams, Communities") at UM2007*, 2007.

[52] Jacques Ferber. *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence.* Addison-Wesley Professional, February 1999.

[53] Eduardo B. Fernandez and Rouyi Pan. A pattern language for security models. In *Proceedings of the 8th Conference on Pattern Languages of Programs (PLoP)*, 2001.

[54] Sally Fincher. PLML: Pattern language markup language. Technical report, Report of Workshop held at CHI, Interfaces, 56 (pp. 26-28), 2003.

[55] Steve Fox, Kuldeep Karnawat, Mark Mydland, Susan Dumais, and Thomas White. Evaluating implicit measures to improve web search. *ACM Trans. Inf. Syst.*, 23(2):147–168, 2005.

[56] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[57] Peter Geczy, Noriaki Izumi, Shotaro Akaho, and Koiti Hasida. Knowledge worker intranet behaviour and usability. *International Journal of Business Intelligence and Data Mining*, pages 447–470, December 2007.

[58] J.L. Gillin and J.P. Gillin. *An Introduction to Sociology.* The Macmillan Company., 1942.

[59] Paulo Gomes, Francisco C. Pereira, Paulo Paiva, Nuno Seco, Paulo Carreiro, Jose L. Ferreira, and Carlos Bento. Using CBR for automation of software design patterns. In *Proceedings of the 6th European Conference on Advances in Case-Based Reasoning (ECCBR'02)*, pages 534–548, London, UK, 2002. Springer-Verlag.

[60] Patricia Gongla and Christine R. Rizzuto. Evolving communities of practice: Ibm global services experience. *IBM Syst. J.*, 40(4):842–862, 2001.

[61] Marco Gori and Ian Witten. The bubble of web visibility. *Communications of the ACM*, 48(3):115–117, 2005.

[62] Stephen Gourlay. Tacit knowledge, tacit knowing or behaving? In *3rd European Organizational Knowledge, Learning, and Capabilities conference, Athens, Greece, 5-6 April*, 2002.

[63] Stephen Gourlay. Tacit knowledge: the variety of meanings in empirical research. In *5th European Organizational Knowledge, Learning and Capabilities conference,; Innsbruck, Austria, 2-3 April*, 2004.

[64] Markus Grabert and Derek Bridge. Case-based reuse of software examplets. *Journal of Universal Computer Science*, 9(7):627–640, 2003.

[65] Sharon L. Greene, Paul M. Matchen, Lauretta Jones, John C. Thomas, and Matt Callery. Tool-based decision support for pattern assisted development. In *CHI 2003 workshop on HCI Patterns: Concepts and Tools*, 2003.

[66] Daniel Gross and Eric S. K. Yu. From non-functional requirements to design through patterns. *Requirements Engineering*, 6(1):18–36, 2001.

[67] M. Hafiz, P. Adamczyk, and R. E. Johnson. Organizing security patterns. *Software, IEEE*, 24(4):52–60, 2007.

[68] Marvin Harris. *Culture, people, nature: An introduction to general anthropology.* New York: Thomas Y. Crowell, 1975.

[69] Richard J. Harrison and Glenn R. Carroll. *Culture and Demography in Organizations*. Princeton University Press, December 2006.

[70] Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, and John T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22(1):5–53, 2004.

[71] Paul M. Hildreth and Chris Kimble. The duality of knowledge. *Information Research*, 8(1), October 2002.

[72] Brian R. Hirshman, Aliaksandr Birukou, Michael K. Martin, Michael W. Bigrigg, and Kathleen M. Carley. The impact of educational interventions on real & stylized cities. Technical Report CMU-ISR-08-114, Carnegie Mellon University, 2008.

[73] Brian R. Hirshman, Kathleen M. Carley, and Michael J. Kowalchuck. Loading networks in construct. Technical Report CMU-ISRI-07-116, Carnegie Mellon University, July 2007.

[74] Brian R. Hirshman, Kathleen M. Carley, and Michael J. Kowalchuck. Specifying agents in construct. Technical Report CMU-ISRI-07-107, Carnegie Mellon University, July 2007.

[75] Bemardo A. Huberman and Michael Kaminsky. Beehive: A system for cooperative filtering and sharing of information. Technical report, Xerox Palo Alto Research Center, 1996.

[76] Hiroshi Ishikawa, Manabu Ohta, Shohei Yokoyama, Takuya Watanabe, and Kaoru Katayama. Active knowledge mining for intelligent web page management. In *Knowledge-Based Intelligent Information and Engineering Systems: 7th International Conference, KES 2003. Proceedings, Part I*, volume 2773 of *Lecture Notes in Computer Science*, pages 975–983, Oxford, UK, 2003. Springer.

[77] Juan, Belén D. Agudo, Derek Bridge, and Pedro. Semantic templates for designing recommender systems. In M. Petridis, editor, *Proceedings of the 12th UK Workshop on Case-Based Reasoning*, pages 64–75. CMS Press, University of Greenwich, 2007.

[78] Roger M. Keesing. *Cultural anthropology: A contemporary perspective*. Holt, Rinehart and Winston, 1981.

[79] U. Keller, R. Lara, and A. Polleres. WSMO web service discovery. Working draft, WSML Working Group, http://www.wsmo.org/2004/d5/d5.1/., 2004.

[80] Elizabeth A. Kendall, Murali P. V. Krishna, Chirag V. Pathak, and C. B. Suresh. Patterns of intelligent and mobile agents. In *AGENTS '98: Proceedings of the second international conference on Autonomous agents*, pages 92–99, New York, NY, USA, 1998. ACM.

[81] M. Kerrigan. Web service selection mechanisms in the web service execution environment (WSMX). In *Proceedings of the ACM Symposium on Applied Computing (SAC)*, pages 1664–1668. ACM Press, 2006.

[82] Alfred Kobsa. Tailoring privacy to users' needs. In *UM '01: Proceedings of the 8th International Conference on User Modeling 2001*, pages 303–313, London, UK, 2001. Springer-Verlag.

[83] N. Kokash, A. Birukou, and V. D'Andrea. Web service discovery based on past user experience. In *International Conference on Business Information Systems (BIS)*, volume 4439 of *LNCS*, pages 95–107. Springer, 2007.

[84] Joseph A. Konstan, Bradley N. Miller, David Maltz, Jonathan L. Herlocker, Lee R. Gordon, and John Riedl. Grouplens: applying collaborative filtering to usenet news. *Commun. ACM*, 40(3):77–87, 1997.

[85] A. L. Kroeber and C. Kluckhohn. Culture: A critical review of concepts and definitions. *Papers of the Peabody Museum*, 47:643–656, 1952.

[86] David C. Kung, Hitesh Bhambhani, Riken Shah, and Gaurav Pancholi. An expert system for suggesting design patterns: a methodology and a prototype. In Taghi M. Khoshgoftaar, editor, *Software Engineering With Computational Intelligence*, Series in Engineering and Computer Science. Kluwer International, 2003.

[87] Yasumasa Kuroda and Tatsuzo Suzuki. Arab students and english: the role of implicit culture. *Behaviormetrica*, 29:23–44, 1991.

[88] Jean Lave and Etienne Wenger. *Situated Learning: Legitimate Periperal Participation*. Cambridge University Press, 1991.

[89] Henry Lieberman. Letizia: An agent that assists web browsing. In Chris S. Mellish, editor, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 924–929, Montreal, Quebec, Canada, 1995. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA.

[90] Greg Linden, Brent Smith, and Jeremy York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, 2003.

[91] Ralph Linton. *The study of man: An introduction*. New York: D. Appleton-Century, 1936.

[92] Zoe Lock and Daniel Kudenko. Interactions between stereotypes. In *AH'06: Proceedings of the 4th International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems*, 2006.

[93] Umardand Shripad Manikrao and T. V. Prabhakar. Dynamic selection of web services with recommendation system. In *Proceedings of the International Conference on Next Generation Web Services Practices (NWESP)*, pages 117–121, Washington, DC, USA, 2005. IEEE Computer Society.

[94] D. Martin, M. Burstein, and et al. OWL-S: Semantic markup for web services. W3c member submission, Web Ontology Working group, http://www.w3.org/Submission/OWL-S/, 2004.

[95] Claudio Masolo, Laure Vieu, Emanuele Bottazzi, Carola Catenacci, Roberta Ferrario, Aldo Gangemi, and Nicola Guarino. Social roles and their descriptions. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference (KR2004)*, Whistler, Canada, 2004.

[96] Paolo Massa and Paolo Avesani. Trust-aware recommender systems. In *RecSys '07: Proceedings of the 2007 ACM conference on Recommender systems*, pages 17–24, New York, NY, USA, 2007. ACM.

[97] Montgomery McFate. The military utility of understanding adversary culture. *Joint Force Quarterly*, 38, July 2005.

[98] Mead. *Cooperation and Competition Among Primitive Peoples*. Transaction Publishers, December 2002.

[99] Margaret Mead and Rhoda Métraux, editors. *The Study of Culture at a Distance*. University of Chicago Press, 1953.

[100] Filippo Menczer. Complementing search engines with online web mining agents. *Decis. Support Syst.*, 35(2):195–212, 2003.

[101] Stuart E. Middleton, Nigel R. Shadbolt, and David C. De Roure. Ontological user profiling in recommender systems. *ACM Trans. Inf. Syst.*, 22(1):54–88, 2004.

[102] CJ. Mimnagh and M. Murphy. Junior doctors working patterns: application of knowledge management theory to junior doctors training. In *Proceedings of the conference on current perspectives in healthcare computing*, pages 42–47, Harrogate, 22-24 March 2004.

[103] Monique B. Mulder, Charles L. Nunn, and Mary C. Towner. Cultural macroevolution and the transmission of traits. *Evolutionary Anthropology: Issues, News, and Reviews*, 15(2):52–64, 2006.

[104] Gunter Mussbacher, Michael Weiss, and Daniel Amyot. Formalizing architectural patterns with the goal-oriented requirement language. In *Proceedings of the Fifth Nordic Pattern Languages of Programs Conference (VikingPLoP'06)*, September 2006.

[105] James Noble. Classifying relationships between object-oriented design patterns. In *Proceedings of the Australian Software Engineering Conference*, pages 98–107. IEEE Computer Society Press, 1998.

[106] Ikujiro Nonaka and Hirotaka Takeuchi. *The Knowledge Creating Company*. Oxford University Press, New York, May 1995.

[107] Andrea Omicini, Alessandro Ricci, and Mirko Viroli. Artifacts in the a&a meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 17(3):432–456, December 2008.

[108] Andrea Omicini, Alessandro Ricci, Mirko Viroli, Cristiano Castelfranchi, and Luca Tummolini. Coordination artifacts: Environment-based coordination for intelligent agents. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 286–293, Washington, DC, USA, 2004. IEEE Computer Society.

[109] Charles O'Reilly. Corporations, culture, and commitment: Motivation and social control in organizations. *California Management Review*, 31(4):9–25, 1989.

[110] Sascha Ossowski. *Co-ordination in Artificial Agent Societies, Social Structure and Its Implications for Autonomous Problem-Solving Agents*, volume 1535 of *Lecture Notes in Computer Science*. Springer, 1999.

[111] Andrei Papliatseyeu. Supporting PCRrelated activity with Implicit Culture. Master's thesis, University of Trento, March 2007.

[112] Patternseer. `http://doc-it.fe.up.pt/aaguiar/space/Projects/PatternSeer`.

[113] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In *CSCW '94: Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pages 175–186, Chapel Hill, North Carolina, United States, 1994. ACM Press.

[114] Alessandro Ricci, Andrea Omicini, Mirko Viroli, Luca Gardelli, and Enrico Oliva. Cognitive stigmergy: Towards a framework based on agents and artifacts. In *Environments for Multi-Agent Systems III*, pages 124–140. Springer, 2007.

[115] Matt Ridley. *Nature Via Nurture: Genes, Experience, and What Makes Us Human*. HarperCollins, May 2003.

[116] Linda Rising. *The Pattern Almanac*. Addison-Wesley Longman Publishing Co., Inc., 2000.

[117] Edgar H. Schein. *Organizational Culture and Leadership (Jossey-Bass Psychology Series)*. Jossey-Bass, 2 edition, December 1996.

[118] Craig Schreiber and Kathleen Carley. The impact of databases on knowledge transfer: Simulation providing theory. In *In Proceedings of NAACSOS Conference*, 2003.

[119] Markus Schumacher. *Security Engineering with Patterns Origins, Theoretical Model, and New Applications*. Number 2754 in LNCS. Springer, 2003.

[120] Ingo Schwab and Alfred Kobsa. Adaptivity through unobstrusive learning. *KI*, 16(3):5–9, 2002.

[121] Nuno Seco, Tony Veale, and Jer Hayes. An intrinsic information content metric for semantic similarity in WordNet. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, pages 1089–1090. IOS Press, 2004.

[122] A. Seffah and H. Javahery. On the usability of usability patterns. In *Workshop Patterns in Practice, CHI*, 2002.

[123] Phoebe Sengers. *Anti-Boxology: Agent Design in Cultural Context.* PhD thesis, Carnegie Mellon University, 1998.

[124] Manu Sharma, Michael Holmes, Juan Santamaria, Arya Irani, Charles Isbell, and Ashwin Ram. Transfer learning in real-time strategy games using hybrid cbr/rl. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 1041–1046. AAAI Press, 2007.

[125] Mildred L. G. Shaw and Brian R. Gaines. Supporting modeling of the social practices of other users in internet communities. In *UM '99: Proceedings of the 7th International Conference on User Modeling*, 1999.

[126] Wanita Sherchan, Seng W. Loke, and Shonali Krishnaswamy. A fuzzy model for reasoning about reputation in web services. In *Proceedings of ACM Symposium on Applied Computing (SAC)*, pages 1886 – 1892. ACM Press, 2006.

[127] Yoav Shoham and Moshe Tennenholtz. Co-learning and the evolution of social activity. Technical report, Stanford university, 1993.

[128] Yoav Shoham and Moshe Tennenholtz. On social laws for artificial agent societies: off-line design. *Artif. Intell.*, 73(1-2):231–252, 1995.

[129] Yoav Shoham and Moshe Tennenholtz. On the emergence of social conventions: modeling, analysis, and simulations. *Artificial Intelligence*, 94(1-2):139–166, July 1997.

[130] Barry Smyth, Evelyn Balfe, Jill Freyne, Peter Briggs, Maurice Coyle, and Oisin Boydell. Exploiting query repetition and regularity in an adaptive community-based web search engine. *User Modeling and User-Adapted Interaction*, 14(5):383–423, 2005.

[131] Gabriel L. Somlo and Adele E. Howe. Using web helper agent profiles in query generation. In *AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 812–818, New York, NY, USA, 2003. ACM.

[132] Ian Sommerville. *Software engineering (7th ed.).* Addison-Wesley, Boston, MA, USA, 2004.

[133] Paolo Alberto Spada. Sistema adattivo per internet radio basato su cultura implicita (in italian). available at `http://disi.unitn.it/~birukou/students/theses/PaoloASpada-BS-2008.pdf`. Bachelor thesis, 2008.

[134] George Spanoudakis and Panos Constantopoulos. Elaborating analogies from conceptual models. *Applied Artificial Intelligence*, 10(4):281–306, 1996.

[135] Dan Sperber. *Explaining Culture: A Naturalistic Approach.* Blackwell Publishers, September 1996.

[136] Dan Sperber and Lawrence A. Hirschfeld. The cognitive foundations of cultural stability and diversity. *Trends in Cognitive Sciences*, 8(1):40–46, January 2004.

[137] M.I. Stadler and P.A. Frensch. *Handbook of Implicit Learning.* Sage Publications, Inc, 1 edition, October 1997.

[138] Lucy A. Suchman. *Plans and Situated Action.* Cambridge University Press, 1987.

[139] Erik Talvitie and Satinder Singh. An experts algorithm for transfer learning. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 1065–1070. AAAI Press, 2007.

[140] Matthew E. Taylor and Peter Stone. Behavior transfer for value-function-based reinforcement learning. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 53–59, New York, NY, USA, 2005. ACM Press.

[141] Roberto Torres, Sean M. McNee, Mara Abel, Joseph A. Konstan, and John Riedl. Enhancing digital libraries with techlens+. In *JCDL '04: Proceedings of the 4th ACM/IEEE-CS joint conference on Digital libraries*, pages 228–236, New York, NY, USA, 2004. ACM Press.

[142] Roy Turner, Elise Turner, Thomas Wagner, Thomas Wheeler, and Nancy Ogle. Using explicit, a priori contextual knowledge in an intelligent web search agent. In *Modeling and Using Context*, pages 343–352. Springer, 2001.

[143] Edward B. Tylor. *Primitive culture.* London: J. Murray, 1871.

[144] Bert van den Berg, Ren van Es, Colin Tattersall, Jos Janssen, Jocelyn Manderveld, Francis Brouns, Hub Kurvers, and Rob Koper. Swarm-based sequencing recommendations in e-learning. In *Proc. of the 5th International Conference on Intelligent Systems Design and Applications (ISDA05)*, 2005.

[145] Bart van den Hooff, Wim Elving, Jan Michiel Meeuwsen, and Claudette Dumoulin. Knowledge sharing in knowledge communities. In Volker Wulf M.H. Huysman, Etienne Wenger, editor, *Communities and technologies*, pages 119–141. Kluwer, B.V., Deventer, The Netherlands, The Netherlands, 2003.

[146] L. Vignollet, M. Plu, J. C. Marty, and L. Agosto. Regulation mechanisms in an open social media using a contact recommender system. In *Proceedings of the Second Communities and Technologies Conference*, pages 419–436, 2005.

[147] Frank Walter, Stefano Battiston, and Frank Schweitzer. A model of a trust-based recommendation system on a social network. *Autonomous Agents and Multi-Agent Systems*, 16(1):57–74, February 2008.

[148] Jing Wang, Yeong-Tae Song, and Lawrence Chung. From software architecture to design patterns: A case study of an nfr approach. In *SNPD-SAWN '05: Proceedings of the Sixth International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing and First ACIS International Workshop on Self-Assembling Wireless Networks*, pages 170–177, Washington, DC, USA, 2005. IEEE Computer Society.

[149] Yan Wei, Nicholas Jennings, Luc Moreau, and Wendy Hall. User evaluation of a market-based recommender system. *Autonomous Agents and Multi-Agent Systems*, 17(2):251–269, October 2008.

[150] Yan Zheng Wei, Luc Moreau, and Nicholas R. Jennings. A market-based approach to recommender systems. *ACM Trans. Inf. Syst.*, 23(3):227–266, 2005.

[151] E.D. Weinberger and S.A. Kauffman. The nk model of rugged fitness landscapes and its application to maturation of the immune response. *Journal of Theoretical Biology*, 141:211–245, 1989.

[152] Etienne Wenger. Communities of practice a brief introduction. available at `http://www.ewenger.com/theory/`.

[153] Etienne Wenger. *Communities of Practice: Learning, Meaning, and Identity*. Cambridge University Press, December 1999.

[154] Richard Wyatt-Haines. *Align IT: Business Impact Through IT*. Wiley, June 2007.

[155] Bin Yu and Munindar P. Singh. An agent-based approach to knowledge management. In *CIKM '02: Proceedings of the eleventh international conference on Information and knowledge management*, pages 642–644, New York, NY, USA, 2002. ACM Press.

[156] Uwe Zdun. Systematic pattern selection using pattern language grammars and design space analysis. *Software: Practice and Experience*, 37(9):983–1016, 2007.

Appendix A

The syntax of the language used for describing actions in the Implicit Culture Framework

In this section, we give the definition of the syntax of the language we use to give examples of actions in the Implicit Culture Framework. We also provide several examples of using the language.

The language L_{SICS} is expressed in the Backus-Naur form. We use <String> to refer to a string, <Integer> to refer to an integer number, and <Real> to refer to a real number.

```
<Timestamp>::=<TemporalVariable>|<TemporalConstant>

<TemporalVariable>::=<Variable>

<TemporalConstant>::=<Integer>

<Variable>::=<String>

<Action>::=<ActionName>(<Agent>;<Objects>;<Attributes>;<Timestamp>)

<ActionName>::=<String>

<Attributes>::= |<Attribute>{,<Attribute>}

<Attribute>::=<AttributeName>=<AttributeVariable>|"<AttributeValue>"

<AttributeName>::=<String>

<AttributeValue>::=<String>|<Integer>|<Real>

<AttributeVariable>::=*|<Variable>
```

```
<Objects>::= |<Object>{,<Object>}

<Object>::=<ObjectInstance>|<ObjectVariable>

<ObjectInstance>::=<ObjectName>[(<Attributes>)]

<ObjectVariable>::=*|<Variable>

<ObjectName>::=<String>

<Agent>::=<AgentInstance>|<AgentVariable>

<AgentVariable>::=*|<Variable>

<AgentInstance>::=<AgentName>[(<Attributes>)]

<AgentName>::=<String>
```

Example 1. The following sentence in the language L_{SICS} says that Peter is a 29 year old and he invoked the operation getWeatherByZip of the DOTSFastWeather web service 25-June-07 at 14:22, using the BSD operating system. Here *Peter* is the name of the agent, *age=29* is the agent attribute, the operation is specified as an object, and the web service is an attribute of the operation. *OS=BSD* is an action attribute, and *25-Jun-07-14:22* is a timestamp of the action.

$$invoke(\text{Peter(age=29); getWeatherByZip(service=DOTSFastWeather); OS=BSD;}$$
$$\text{25-Jun-07-14:22})$$

The following sentence illustrate using a wildcard to say that the action is performed by an agent who is 29 year old.

$$invoke(\text{*(age=29); getWeatherByZip(service=DOTSFastWeather); OS=BSD;}$$
$$\text{25-Jun-07-14:22})$$

⊘

Example 2. This example illustrates the use of variables in the cultural theory. The theory contains two actions described in L_{SICS}. The first action says that someone, at some point of time, requests [pattern] for the problem $\_x$ described by the attribute *keyword = $\_y$*. The second action says that someone, at some point of time, applies some pattern (specified with the attribute *pattern_name* = *) for the problem $\_x$. Note that $\_x$ and $\_y$ are variables (start with '_') and not constants. If $\_y$ in the antecedent is substituted with some value, e.g. "single access point", then $\_y$ in the consequent is also substituted with the same value.

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

⊘

Appendix B

The list of publications

Here is the list of publications derived from the content of the thesis. Please see `http://disi.unitn.it/~birukou/index.php?page=publications` for the complete list.

Journal papers

1. A. Birukou, E. Blanzieri, V. D'Andrea, P. Giorgini, N. Kokash. Improving Web Service Discovery with Usage Data. IEEE Software, 24(6): 47-54, 2007. Special issue on Realizing Service-Centric Software Systems.

Conference papers

1. A. Birukou, M. Weiss. A service for software pattern selection. Submitted to the 14th European Conference on Pattern Languages of Programs (EuroPLoP2009), Irsee Monastery, Bavaria, Germany, July 8-12, 2009.

2. A. Birukou, E. Blanzieri. A model of culture for software. Submitted to the 1st and 17th International Conference on User Modeling, Adaptation, and Personalization (UMAP2009), Trento, Italy, June 22-26, 2009.

3. N. Kokash, A. Birukou, V. D'Andrea. Web Service Discovery Based on Past User Experience. Proceedings of the 10th International Conference on Business Information Systems (BIS2007), Poznan, Poland, April 25-27, 2007.

4. A. Birukou, E. Blanzieri, V. D'Andrea, P. Giorgini, N. Kokash, A. Modena. IC-Service: A Service-Oriented Approach to the Development of Recommendation Systems. Proceedings of ACM Symposium on Applied Computing. Special Track on Web Technologies (SAC2007), Seoul, Korea, March 11-15, 2007.

5. A. Birukou, E. Blanzieri and P. Giorgini. Choosing the Right Design Pattern: the Implicit Culture Approach. Workshop on Multi-Agent Systems and Simulations at the 4th Industrial Simulation Conference (ISC2006), Palermo, Italy, June 5-7, 2006.

6. A. Birukou, E. Blanzieri and P. Giorgini. A Multi-Agent System that Facilitates Scientific Publications Search. In Proceedings of the 5th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS2006), Hakodate, Hokkaido, Japan, May 8-12, 2006.

7. A. Birukou, E. Blanzieri and P. Giorgini. Implicit: An agent-based recommendation system for web search. In Proceedings of the 4th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS2005), Utrecht, the Nederlands, July 25-29, 2005.

Workshop papers

1. M. Weiss, A. Birukou. Building a Pattern Repository: Benefiting from the Open, Lightweight, and Participative Nature of Wikis. Workshop on Wikis for Software Engineering at ACM WikiSym, 2007 International Symposium on Wikis (WikiSym), Montre'al, Que'bec, Canada, October 21-23, 2007.

2. A. Birukou, E. Blanzieri and P. Giorgini. Implicit Culture as a Tool for Social Navigation . Workshop on the Social Navigation and Community-Based Adaptation Technologies at the International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems (AH2006), Dublin, Ireland, June 20-23, 2006.

3. A. Birukou, E. Blanzieri and P. Giorgini. Implicit: A recommender system that uses implicit knowledge to produce suggestions. Workshop on Multi-Agent Information Retrieval and Recommender Systems at the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05), Edinburgh, Scotland, July 30-August 5, 2005.