Social Computing—Bridging the Gap Between the Social and the Technical

Christoph Beckmann, Tom Gross

Human-Computer Interaction Group, University of Bamberg, Germany
(<fistname>.<lastname>(at)uni-bamberg.de)

Abstract. Developing cooperative systems and social media requires taking complex decisions concerning the social interaction to be supported as well as the technical foundation. In this paper we build on the long and successful tradition of design patterns and the social framework of Erving Goffman. We present design patterns that address both challenges of social interaction and technical foundation—they provide input for software developers with respect to structuring software and to providing adequate support for the interaction of users with the environment and with each other.

Keywords. Social Computing; Software Design; Cooperative Systems; Social Media.

1 Introduction

In social computing, systems aim at facilitating communication and cooperation among users who are either at the same location or at different locations. Social media summarises concepts and systems that aim at an active participation of users during an interaction, easy exchange of information, and sophisticated self-presentation [11].

Developing concepts for those systems is a challenging task and has been researched for more than two decades [12] [7]. They often have a strong influence on the structure and flow of the interaction in the group, as Schmidt [18, p. vii] explains: 'the development of computing technologies have from the very beginning been tightly interwoven with the development of cooperative work'. And he [18, p. vii] continues: 'our understanding of the coordinative practices, for which these coordination technologies are being developed, is quite deficient, leaving systems designers and software engineers to base their system designs on rudimentary technologies. The result is that these vitally important systems, though technically sound, typically are experienced as cumbersome, inefficient, rigid, crude'.

Patterns have a long and successful tradition for drafting, for documenting, and for reusing the underlying concepts. Very prominently, Christopher Alexander has suggested and provided design patterns in architecture [2]. He introduced a pattern language to describe solutions that were repeatedly applied to reoccurring challenges in the design of buildings. In software engineering software design patterns have been successfully used for documenting and reusing knowledge and provided a 'way of supporting object-oriented design' [20, p. 422]. With respect to social computing, design patterns can document the knowledge and experience with developing cooperative technology.

All these different types of design patterns provide valuable input for cooperative systems and social media. However, there are also limiting factors: software design patterns primarily help structuring software, and cooperative design patterns are primarily based on the analysis of existing cooperative systems or on some specific ethnographic studies. Therefore, the gap is that the complex task of making both types of patterns compatible is in the hands of software designers and developers.

In this paper we build on the history of patterns and present overarching design patterns for social computing systems. For this purpose we leverage on the works of Erving Goffman who studied social interaction among humans and their use of their technical environment for several decades and derived a framework for social interaction. He uses a metaphor of a performance where everybody is an actor that present her- or himself and acts with others.

In the next section we provide a background of patterns. We then introduce the framework of social interaction of Erving Goffman. We discuss how this framework informs the design of cooperative systems and we derive design patterns for cooperative systems that are modelled in a unified modelling language format for software designers and developers. Finally, we summarise our contribution.

2 Background of Design Pattern

Christopher Alexander et al. [3] were the first to systematically distil patterns from reoccurring solutions to reoccurring problems. In the domain of architecture they identified a language of connected patterns for designing buildings. In this section we introduce patterns related to the design of software in general as well as for cooperative systems and social media in particular that build of Alexander et al.

2.1 Software Design Patterns

Software designers and developers widely use software design patterns. Gamma, Helm, Johnson and Vlissides [5] suggested the most notable pattern language for object-oriented software development. They characterise a pattern as a composition of a problem that during the development frequently occurs, a principal solution to the problem, and consequences from applying the solution. Their pattern language includes 23 patterns for classes (i.e., static relationships during compile-time) and objects (i.e., dynamic relationships during run-time) in three categories: creational, structural, and behavioural patterns [21].

Cooperative systems and social media use network-based and distributed software architectures in the background. POSA2 offers a rather technical pattern language addressing the challenges of distributed software architectures especially in the context of object-oriented middleware such as CORBA, COM+, or Jini [17]. It has four categories representing the main challenges of object-oriented middleware: 'Service Access and Configuration', 'Event Handling', 'Concurrency', and

‘Synchronisation’. The description of patterns is extensive and contains precise design implication for the named middleware along verbose source code examples.

While software design patterns are substantial for sustainable software development, they still leave the burden of the complexity of social interactions to software designers and developers of cooperative systems.

2.2 Design Patterns for Cooperative Systems and Social Media

Design patterns for cooperative systems and social media typically focus on human behaviour and interaction. We describe patterns that support designers and developers of cooperative systems and social media.

A pattern language for computer-mediated interaction condenses features and properties of existing cooperative systems [19]. It has three categories: ‘community support’, ‘group support’, and ‘base technology’. The variety of patterns reaches from simple ones (e.g., the ‘login pattern’ allows users to interact within a system as individuals with an associated user accounts) to complex ones (e.g., the ‘remote field of vision pattern’ allows users, which work remotely on shared artefacts to be aware of at which parts others are currently watching at). The description of patterns is very detailed and considers caveats as well as implications for security.

Specific patterns for privacy and sharing provide solutions to problems concerning the quality of use of cooperative systems [4]. They result from field studies, notes, and design sketches that were translated into three patterns: The ‘workspace with privacy gradient’, the ‘combination of personal and shared devices’, and the ‘drop connector’.

Descriptive patterns have been suggested to allows a better facilitation of the communication in interdisciplinary design teams during the development process [13, 14]. They are comprehensive and express ‘generally recurrent phenomena’ extracted from ethnographic studies at workplaces. The resulting descriptive pattern language consists of six patterns: ‘multiple representations of information’, ‘artefact as an audit trail’, ‘use of a public artefact’, ‘accounting for an unseen artefact’, ‘working with interruptions’, and ‘forms of co-located teamwork’. Their patterns are extracted from fieldwork results using two types of properties: ‘spatially-oriented features’ and ‘work-oriented features’. Their patterns can be extended with a ‘vignette’, which describes real examples as special use cases and provide further design implications.

Despite the fact that patterns for cooperative systems and social media provide detailed insights into practices and requirements of users working together, they mostly lack the dynamic notion of such systems, where users can take advantage of a throughout personalisation of their environment.

3. Goffman’s Framework of Social Interaction

We introduce the background and major concepts of Goffman’s framework of social interaction that are relevant for designers and developers of cooperative systems. Goffman [6] studied social interaction among humans for several decades and developed a conceptual framework of social interaction among humans in face-to-face situations. It is based on his own observations, on observations of other researchers, and on informal sources. In the following, we describe Goffman's framework in three categories: participants, regions, and performance (cf. **Fig. 1**).

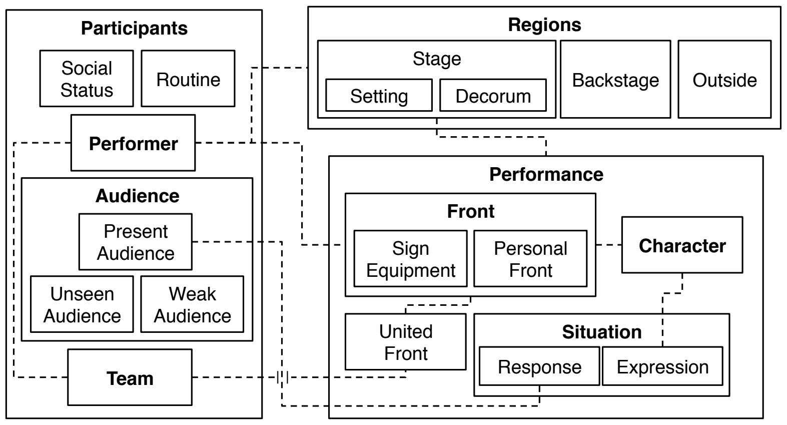


Fig. 1. Major Concepts in Goffman's framework of social interaction.

3.1 Participants

Participants act according to their social status (i.e., socio-economic standing in the society). They present a routine (i.e., a 'pre-established pattern of action which is unfolded during a performance' [6, p. 16]). For Goffman humans follow two types of ideals when interacting with each other: the optimistic ideal of full harmony, which according to Goffman is hard to achieve; and the pragmatic ideal as a projection that should be in accordance with reality and that others can accept—at least temporarily—without showing deep and inner feelings of the self.

In a performance a performer and an audience are involved. A performer defines a situation through a projection of reality as expressions of a character bound to a certain social role in front of an audience. Performers anticipate their audience and continuously adapt their performance according to its responses. Goffman distinguished three audience types. The present audience attends the performance, receives expressions, verifies them according to the projected situation, and responds. The unseen audience is imaginary and used to anticipate a performance. The week audience is real, but not present. It constitutes of other performers giving similar performances. In preparation of a performance, a performer can exchange experiences and responses with the weak audience to improve her own ability to be convincing.

Goffman describes the collaboration of performers as a 'performance team'. Its members ideally fit together as a whole in presenting similar individual performances to amplify a desired projection, or in presenting dissimilar performances that complement to a joint projection.

3.2 Regions

Regions are spatial arrangements used for performances and include specific media for communication as well as boundaries for perception. Goffman names three types of regions: stage, backstage, and outside. A stage provides a setting for the actual performance and is embroidered with decorative properties (i.e., decorum). It supports performers in fostering a situation. Both the performers, as well as the audience can access the stage, having different perspectives. The backstage is a region that performers can access to prepare and evaluate their performance. Also team members suspend backstage. The audience cannot access the backstage. The outside region describes the third type that is neither stage nor backstage. Although it will be excluded from a performance, performers will prepare and use a dedicated front for the outside (e.g., the façade of buildings of a company).

3.3 Performance

For Goffman a performance means social interaction as a finite cycle of expressions to define a situation and of responses as feedback of validity. A performance takes place in a region of type stage. For a performance, each performer prepares a set of fronts, which represents her towards the audience. A front unites material and immaterial parts. Sign equipment is a front's material part and denotes to all properties required to give a convincing performance. The personal front is a front's immaterial part and denotes to certain types of behaviour of a performer (e.g., speech patterns). It combines 'appearance' (i.e., presenting a performer's social status) and 'manner' (i.e., presenting a performer's interactional role).

Characters make the appearance of performers on the stage. A character—as a figure—is composed of a 'front', which is specifically adapted to the audience and performance. In a performance team, the team as whole has a united front (e.g., according to a professional status) and each member has a character with an associated front to invoke during staging. During a performance a character plays routines to convey acceptable and to conceal inacceptable expressions. In a performance team multiple characters will follow this behaviour.

Expressions are information that is communicated by a character using 'sign-vehicles' (i.e., information carriers). There are wanted expressions that are acceptable and foster a situation as a valid projection of reality, and unwanted expressions that are inacceptable and inappropriate for a given performance in front of a particular audience. In order to manifest a performance that is coherent, a performer strives to communicate expressions consistently through their characters towards an audience. Thus a performer's character endeavours to conceal unwanted expressions.

Responses are all kinds of feedback. An audience continuously verifies the performance according to the defined situation and the overall reality as well as to the front of the character. It responds the result to the performer.

In order to manifest a valid performance, performer and audience agree on three principal constructs that prevent a false or doubtful projection of reality based on contradictive expressions or discrediting actions: The 'Working Consensus' is an agreement on the definition of the situation and describes a temporal value system among all participants. The ‘Reciprocity’ means that performers guise their characters to act according to the situation (i.e., provoke neither intentionally nor factually misunderstandings) and that the audience responds to performance according to the situation (i.e., allege neither consciously nor unconsciously false behaviour). The ‘Interactional Modus Vivendi’ describes that an individual in the audience only responds to expressions that are important for the individual; the individual in the audience remains silent in things that are only important to others.

Goffman describes additional participants. For instance, the team support, which is one of the following: colleagues that constitute the weak audience, training specialists that build up a desirable performance, service specialists that maintain a performance, confidants that listen to a performer’s sins, or renegades that preserve a idealistic moral stand that a performer or team failed to keep. Goffman also defines outsiders as being neither performers nor audience having little or no knowledge of the performance. They can access the type outside region.

4 Informing the Design of Social Computing

In this section we transform Goffman’s framework into design patterns. We used three steps. We first identified key statements of Goffman’s framework concerning structural aspects (i.e., social entities involved into interactions) and dynamic aspects (i.e., actions of and interactions between social entities). In a second step, we augmented these aspects with literature reviews and lessons learned from conceptualising and developing cooperative systems—especially concerning the transition from physically co-present humans to virtually co-present humans (e.g., [8, 9]). In a third step, we iteratively derived four design patterns for cooperative systems and modelled them in the unified modelling language (UML version 2.4 [16]).

4.1 Structure of Social Computing

The structure of social computing systems refers to entities and their relations as essential ingredients. Our UML class diagrams emphasise entities involved, their compositions, and their dependencies. We use interfaces for modelling general entity behaviour that can be applied to a variety of instances. We use abstract classes for modelling entities that share implementations, and we use standard classes for modelling specific entities.

The first structural pattern we introduce is the *Social Entity Pattern* (cf. **Fig. 2**). It describes the general setting of people involved in an interaction and their roles. The interface *SocialEntities* refers to humans that are explicitly included in an interaction. A social entity has general knowledge of the world and specific knowledge of particular domains. It relies on *Routines* as ‘pre-established pattern of action […] which may be presented or played…’ [6, p. 16]. It conveys information it likes to share with others, and conceals information it likes to hide from others. There are four classes implementing the interface *SocialEntity*: *ActiveIndividuals*, *ActiveTeams, PassiveIndividuals*, and *PassiveTeams*.

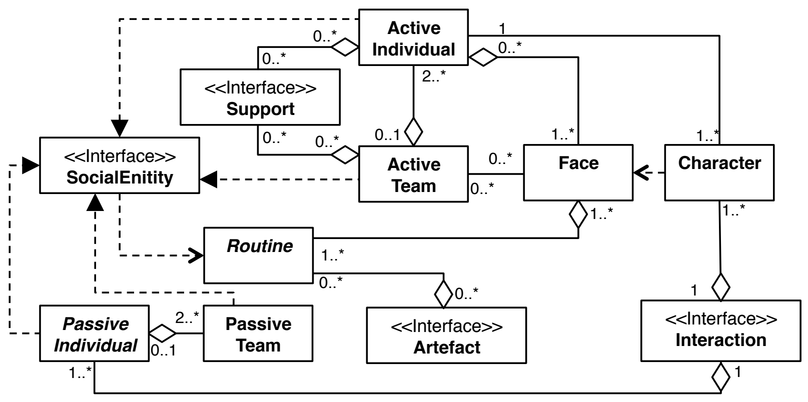


Fig. 2. Social Entity Pattern as UML class diagram.

ActiveIndividuals refer to Goffman's performers and are instances of classes with a repertoire of *Faces*. They anticipate the behaviours of others and select as well as fit their faces towards them. An *ActiveTeam* consists of at least two *ActiveIndividuals*: which refers to 'any set of individuals who cooperate in staging a single routine' and '…an emergent team impression arises which can conveniently be treated as a fact in its own right…' [6, p. 79]. Teams have an overall goal. As noted above members of a team can have in individual activity or a shared activity. Since the delegation of an *ActiveTeam's* members can vary from team to team, it is the responsibility of extended classes to implement that behaviour. An *ActiveIndividual* and *ActiveTeam* can rely on their *Support* (i.e., social entities that provide services or feedback).

PassiveIndividuals as an abstract class implements the interface social entity with the ability to observe an action. Further implementations of such passive individuals are the *PassiveTeam*, which refers to the audience that participates in the interaction. About the relationship of active individuals and passive individuals Goffman states: '…the part one individual plays is tailored to the parts played by the others present, and yet these others also constitute the audience' [6, p. xi]. Parallel to the team above a *PassiveTeam* is an aggregation of *PassiveIndividuals*; Goffman writes: 'There will be a team of persons whose activity … in conjunction with available props will constitute the scene from which the performed character's self will emerge, and another team, the audience.' [6, p. 253].

In the pattern a *Face* class lays out the foundation for a distinct configuration of an active individual or team as a prototype to be applied in an interaction. Our notion of a face refers to Goffman's front; it is the 'part of the individual's performance which regularly functions in a general and fixed fashion to define the situation for those who observe the performance' [6, p. 22]. An *ActiveIndividual* can have multiple faces as a repository of communication methods and properties towards passive individuals. Since, in cooperative systems simultaneous interactions are likely, it is important to note that an active individual may have multiple active faces at a time (i.e., a system is required to provide means for the preparation of an interaction as well as means for easy access to the repository of faces to choose from).

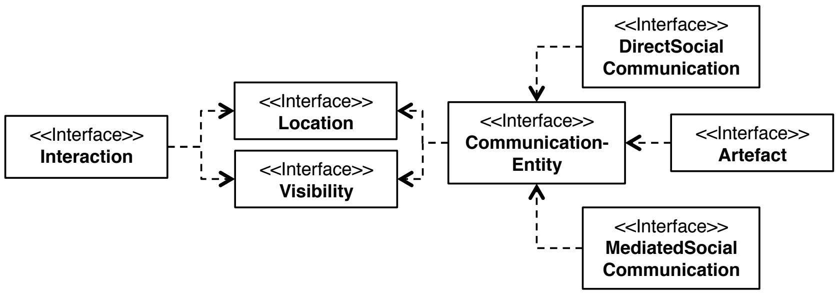
A *Character* is a specific configuration of a face. When instantiated in an interaction, an *ActiveIndividual* selects and transforms a face into a *Character* containing information and dissemination methods: ‘When a participant conveys something during interaction, we expect him to communicate only through the lips of the character he has chosen to project’ [6, p. 176].

In our pattern the interface *Artefact* refers to work-related (e.g., documents) and leisure-related objects (e.g., movies). In contrast, Goffman narrows the performance down to interacting individuals or teams; for Goffman external objects contribute to the overall expression of a situation as a setting: ‘there is the setting, involving furniture, decor, physical layout, and other background items which supply the scenery and stage props for the spate of human action played out before, within, or upon it.’ [6, p. 22]. However, in social computing often an *Artefact* is an essential part of an interaction. It relates to virtual or physical objects that can be created, edited, and deleted in the course of an interaction.

In a routine, a composition of artefacts that can be involved; in social computing systems this is typically represented as collaborative editing or sharing.

The *Interaction* refers to Goffman’s performance. It is a composition of characters of one or more active and one or more passive individuals. It has three phases: in the preparation an active individual sets her role; in the execution a character acts towards passive individuals or a passive team; and in the finalisation an active individual collects responses from its interaction and uses the outcome for further refinements of its faces. A history as set of interactions is important in social computing systems for verifying information and deducing information (i.e., drawing conclusions).

The second structural pattern is the *Region Pattern* (cf. **Fig. 3**). It maps Goffman’s regions into a combination of *Visibility* and *Locality* that can be applied in *Interactions*. Goffman writes: ‘A region may be defined as any place that is bounded to some degree by barriers to perception.’ [6, p. 106]. As described above, Goffman distinguishes the regions stage, backstage, and outside. However, in our opinion, social computing systems require a more flexible representation that should allow for and contribute to in-between regions.



<<Interface>> DirectSocial Communication
<<Interface>> Interaction
<<Interface>> Location
<<Interface>> Visibility
<<Interface>> Communication-Entity
<<Interface>> Artefact
<<Interface>> MediatedSocial Communication

Fig. 3. Region Pattern as UML class diagram.

The interface *Visibility* represents filters for types of information and dissemination methods to be applied to interaction with social entities. While active individuals and active teams can access a huge amount of information, passive individuals and passive teams can only access designated information.

The interface *Locality* also refers to filters, but they provide methods as a boundary of real locations (e.g., a display in a shared office space) or virtual locations (e.g. a user's timeline). Combining *Visibility* and *Locality* provides means for sophisticated configurations than the region types proposed by Goffman could cover. The combination reflects individual sharing preferences that also apply during a system's automatic inference of information (i.e., map and reduce). For instance, an interaction can span real and virtual locations at once while communication is still filtered. The filtering can be achieved by matching properties of *CommunicationEntities* (e.g., an *Artefact* as a shared object) towards the properties of passive entities involved into the interaction. Subsequently, we introduce the interfaces *DirectSocialCommunication* and *MediatedSocialCommunication* along their patterns.

4.2 Dynamics of Social Computing

The dynamics of social computing systems refers to the general communication behaviour of humans within the system. The two patterns focus on the interaction between an *ActiveIndividual* and a *PassiveTeam* as the execution of an interaction.

We show two patterns as use cases in UML sequence diagrams. Each diagram shows the entities involved in the execution, and sequences of synchronous and asynchronous calls used in it (please note, we explain an interaction of an individual, for team performances the steps are similar).

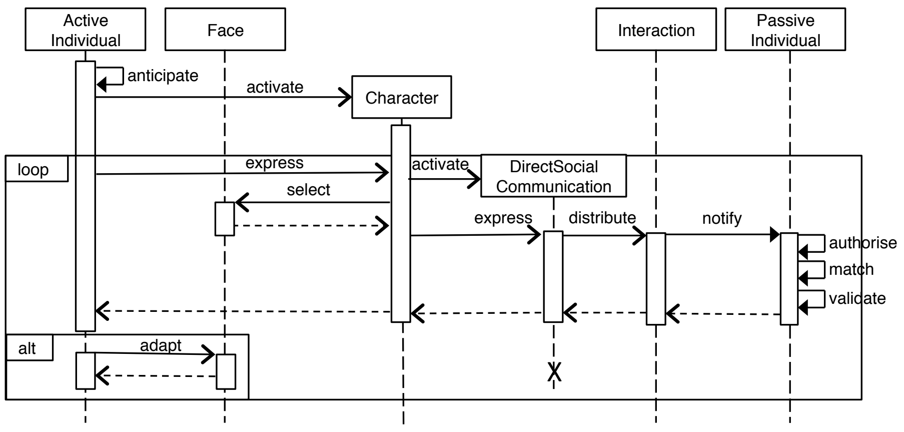


Fig. 4. Direct Social Communication Pattern as UML sequence diagram.

The first dynamic pattern is the *Direct Social Interaction Pattern* (cf. **Fig. 4**). It starts with the path an *ActiveIndividual* executed to setup its *Face* and *Character* and activates a *PassiveIndividual*—summarised as anticipate-call in the diagram. After that, an *ActiveIndividual* instantiates a *Character* object for direct social interaction.

According to Goffman, faces are selected and adapted, rather than created; he writes: 'different routines may employ the same front, it is to be noted that a given social front tends to become institutionalised in terms of the abstract stereotyped expectations to which it gives rise, and tends to take on a meaning and stability apart from the specific tasks which happen at the time' [6, p. 27]. This manner of stereotypical selection and adaptation allows *PassiveIndividuals* to recognise familiarity between *Characters* of different *ActiveIndividuals* and thus simplifies the validation process.

The *Character* object creates the *DirectSocialCommunication* object for delivering information. In the loop of direct social communication, a *Character* calls its associated *Face* to obtain valid and appropriate information. It then delegates this information to the *DirectSocialCommunication* object for further distribution in an *Interaction*. Goffman describes direct social interaction as a communication of 'sign-activity'—the transmission of expressions towards the audience relying on 'sign-vehicles'. He distinguishes two 'radically different' types of communication: the given and the given-off [6, p. 2]. In this pattern *DirectSocialCommunication* refers to the type 'given'. It stands for communication in a narrow sense as it consists of verbal or written symbols (i.e., speech and text). All social entities involved in an interaction are familiar with the encoding and decoding these symbols.

The process of delivering *DirectSocialCommunication* occurs frequently in a loop and simultaneously during an interaction, the resulting calls are asynchronous ones. As described previously, a *PassiveIndividual* receives the information and matches its consistency. A *PassiveIndividual* responds accordingly concerning the information's inner validity (e.g., authorisation of sender and contents) as well as regarding previously received ones (e.g., history of the interaction).

An *ActiveIndividual* can emphasise information during sending *DirectSocialCommunication* as it can adapt a *Face* using the responses received—Goffman speaks of governable aspects [6, p. 7].

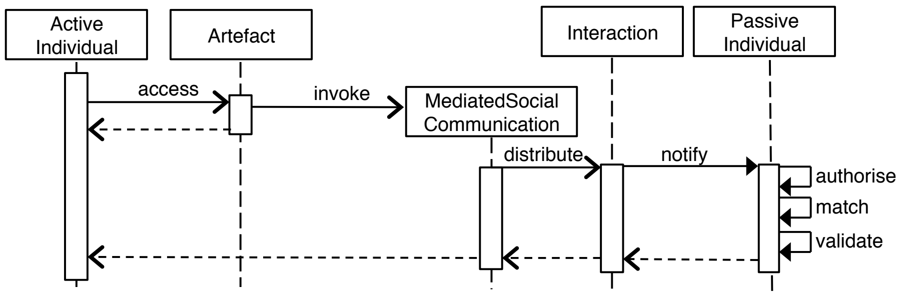


Fig. 5. Mediated Social Interaction Pattern as UML sequence diagram.

The second dynamic pattern we introduce is the *Mediated Social Interaction Pattern* (cf. **Fig. 5**). It reflects the process of accessing an artefact and distributing occurring information of accessing it towards the passive individuals. In cooperative systems and social media applications this type of information is typically used for providing awareness information to the users [10].

MediatedSocialCommunication refers to communication in a broader sense and is related to Goffman's 'given-off'. It consists of a range of behaviours that can hardly controlled or manipulated—Goffman writes of ungovernable aspects and is of 'more theatrical and contextual kind, the non-verbal, presumably unintentional kind, whether this communication be purposely engineered or not.' [6, p. 4]. When accessing an *Artefact* at least one separate object *MediatedSocialCommunication* is created automatically. The *PassiveIndividual* receives information and matches it with previous objects of the type *DirectSocialCommunication* and *MediatedSocialCommunication* and responds accordingly.

5 Discussion and Conclusions

In this paper, we have argued that designers and developers of social computing systems face complex design decisions. To support them, we identified key concepts of Goffman's framework and derived structural and dynamic UML patterns.

Our study of Goffman's framework and the derived patterns relate to some findings of previous work on patterns—corroborating these findings. Our patterns bridge between the artefact-specific patterns of Martin et al. [13, 14] and the collaboration-specific patterns of [19]. The *Social Entity Pattern* represents the typical behaviour of users frequently switching their hats between the two roles of an active individual and a passive individual. The faces they rely on during their performances are diverse in terms of contained information, actions, and reactions. System should address this need for diversity by providing a repository of faces the users can chose from and evolve their characters upon. Yet, our pattern reaches beyond the existing ones, as it allows multiple, persistent, temporal and spatial active characters. The *Region Pattern* addresses the requirement of diverse spaces for preparing, sharing, and acting. Social computing systems should provide these spaces, as users need them for their performance. On the one hand users prepare the interaction using more 'technical standards' in a 'backstage' region where 'the suppressed facts make an appearance.' [6, p. 112]. On the other hand, users interact on 'stage' type regions using more 'expressive standards'. Providing stability of locality and visibility in this pattern is important for preventing users of unmeant disclosures that Goffman calls 'some major forms of performance disruption—unmeant gestures, inopportune intrusions, faux pas, and scenes.' and 'When these flusterings or symptoms of embarrassment become perceived, the reality that is supported by the performance is likely to be further jeopardised and weakened' [6, p. 212]. The *Direct Social Interaction Pattern* and *Mediated Social Interaction Pattern* cope with the performance itself and provide a model for Goffman's two communication types of 'given' and 'given-off'. Users require means of dramaturgical discipline—for instance, the anticipation of the passive individuals—to manage their impression validly. The patterns explicitly inform designers and software engineers of social computing systems to apply the *Region Pattern* in order to consider the hardly to governable type of communication (e.g., when accessing resources within the system or generating meta data).

For future work the structural and the dynamic patterns should be applied in the design of social computing systems so their actual benefit for designers and developers in conceptualising and implementing can be measured in empirical studies. Furthermore, Goffman offers detailed descriptions of more social processes (e.g., make work) and best practices (e.g., team collusion) that may supply further patterns towards an extensive language of patterns for social computing systems.

References

1. Alexander, C. The Timeless Way of Building. Oxford University Press, N.Y., 1979.
2. Alexander, C., Ishikawa, S. and Silverstein, M. A Pattern Language: Towns, Buildings, Construction. Oxford University Press, Oxford, 1977.
3. Alexander, C., Isikawa, S. and Silverstein, M. A Pattern Language. Oxford University Press, Oxford, 1977.
4. Arvola, M. Interaction Design Patterns for Computers in Sociable Use. Int. J. of Computer Applications in Technology 25, 2/3 (Feb. 2006). pp. 128-139.
5. Gamma, E., Helm, R., Johnson, R. and Vlissides, J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, 1994.
6. Goffman, E. The Presentation of Self in Everyday Life. Doubleday Anchor Books, N.Y., 1959.
7. Gross, T. Supporting Effortless Coordination: 25 Years of Awareness Research. Computer Supported Cooperative Work: The J. of Collaborative Computing, Springer-Verlag 22, 4-6 (Aug.-Dec. 2013). pp. 425-474.
8. Gross, T. and Beckmann, C. Advanced Publish and Subscribe for Distributed Sensor-Based Infrastructures: The CoLocScribe Cooperative Media Space. In Proc. of the Seventeenth Conf. on Parallel, Distributed, and Network-Based Processing - PDP 2009. IEEE CS Press, Los Alamitos, 2009. pp. 333-340.
9. Gross, T. and Oemig, C. From PRIMI to PRIMIFaces: Technical Concepts for Selective Information Disclosure. In Proc. of the 32nd Conf. on Software Engineering and Advanced Applications - SEAA 2006. IEEE CS Press, Los Alamitos, 2006. pp. 480-487.
10. Gross, T., Stary, C. and Totter, A. User-Centered Awareness in Computer-Supported Cooperative Work-Systems: Structured Embedding of Findings from Social Sciences. Int. J. of Human-Computer Interaction 18, 3 (Nov. 2005). pp. 323-360.
11. Kaplan, A.M. and Haenleina, M. Users of the World, Unite! The Challenges and Opportunities of Social Media. Business Horizons 53, 1 (Jan.-Feb. 2010). pp. 59-68.
12. Marca, D. and Bock, G., eds. Groupware: Software for Computer-Supported Cooperative Work. IEEE CS Press, Los Alamitos, 1992.
13. Martin, D., Rodden, T., Rouncefield, M., Sommerville, I. and Viller, S. Finding Patterns in the Fieldwork. In Proc. of the Seventh European Conf. on Computer Supported Cooperative Work - ECSCW 2001. Kluwer Academic Publishers, Dortrecht, 2001. pp. 39-58.
14. Martin, D., Rouncefield, M. and Sommerville, I. Applying Patterns of Cooperative Interaction to Work (Re)Design: E-Government and Planning. In Proc. of the SIGCHI Conf. on Human Factors in Computing Systems - CHI 2002. ACM, N.Y., 2002. pp. 235-242.
15. Martin, D. and Sommerville, I. Patterns of Cooperative Interaction: Linking Ethnomethodology and Design. ACM Trans. on Computer-Human Interaction 11, 1 (Mar. 2004). pp. 59-89.
16. Object Management Group Inc. Documents Associated With Unified Modelling Language (UML), V2.4. http://www.omg.org/spec/UML/2.4/, 2011. (Accessed 5/2/2014).
17. Schmidt, D.C., Stal, M., Rohnert, H. and Buschmann, F. Pattern-Oriented Software Architecture. John Wiley & Sons, Chichester, 2000.
18. Schmidt, K. Cooperative Work and Coordinative Practices - Contributions to the Conceptual Foundations of Computer-Supported Cooperative Work (CSCW). Springer-Verlag, Heidelberg, 2011.
19. Schuemmer, T. and Lukosch, S. Patterns for Computer-Mediated Interaction. John Wiley & Sons, Chichester, 2007.
20. Sommerville, I. Software Engineering 8. Pearson Education Limited, Harlow, 2007.
21. Stevens, W.P., Myers, G.J. and Contantine, L.L. Structured Design. IBM Systems J. 13, 2 (1974). pp. 115–139.