PATTERN-BASED COLLABORATION IN AD-HOC TEAMS THROUGH MESSAGE ANNOTATION

Daniel Schall, Robert Gombotz, Schahram Dustdar
Distributed Systems Group, Institute of Information Systems,
Vienna University of Technology, Argentinierstrasse 8, 1040 Wien, Austria
{schall, gombotz, dustdar}@infosys.tuwien.ac.at

Keywords: Collaboration Patterns, Message Annotation, Coordination Support, Interaction Patterns

Abstract: In this paper we present a specification for annotating messages to enable computer-supported message processing, addressing, and analysis. The benefits of annotating messages according to our XML based specification are two-fold: Firstly, it allows computer support during collaboration by enabling automated message addressing (i.e., determining who should get a message) and message management (e.g., managing your messages according to activities, projects, and tasks). Secondly, it enables post-collaboration analysis of messages and mining of message logs for patterns and for workflow models. We provide a proof of concept by presenting how annotated messages may support and facilitate collaboration that happens according to certain collaboration patterns. In addition to the patterns we have already introduced in our previous work, we present more patterns such as *Monitors* that emphasize the applicability of computer supported message handling.

1 INTRODUCTION

In our previous work we have made the case for human interaction patterns in collaborative working environments. A lot of communication in human collaboration is message based.

The problem when trying to mine message logs for pattern or workflow information is that in many cases raw messages can (a posteriori) not be mapped to a context or an activity in an automatic way. We believe that the mining of a message's content for such information (i.e., through text mining technologies) has not yet matured to a stage where it can provide results that satisfy our needs in terms of quality and reliability.

The shortcomings of *automatic message interpretation* in terms of context and activity also become apparent when attempting to provide computer support for message management in terms of relevance and prioritization (e.g., who should receive a message and with what level of urgency).

Message-based collaboration highly relies on the user being responsible for reading, interpreting, and processing messages while very little support is provided by messaging technologies. We witness this every day when masses of *spam* messages are delivered to our inboxes along with those messages that are actually relevant to us. And within the relevant messages there is little to no support for ranking or ordering messages according to urgency, relevance, or context.

2 COLLABORATION PATTERNS

The research conducted in our group aims at developing a pattern language describing the structure, dynamics, and the interaction flows observed in human collaboration. In our earlier work we have presented three initial patterns found in software engineering and applied them to the domain of CWE (Collaborative Working Environments). (Dustdar and Hoffmann, 2006), (Gombotz et al., 2006). These patterns were the Broker, the Proxy, and the Master/Slave pattern which in human collaboration can be interpreted as receptionist, a secretary, and boss/assistant, respectively. As an example, Figure 1 depicts a

Master/Slave pattern. An Initiator I starts an interaction by sending a request to the Receiver R (the Master in this pattern). R delegates sub-tasks to the Contributors C. Note that the interaction between R and C may not be visible to I.

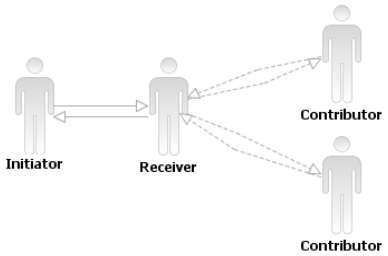

Figure 1: Delegation of sub-tasks (Master/Slave style).

In the following subsections we introduce a new pattern in human collaboration which we termed *Monitor*. The pattern is then further refined into four subtypes based on the monitor's involvement and role in a collaboration.

2.1 Monitor Pattern

Monitoring is omnipresent in collaborative environments and may even happen subconsciously. It is done for different reasons and is achieved through different methods which are discussed in more detail along with the corresponding subtypes of a Monitor in the following subsections.

We define a Monitor as an actor watching or observing a given object of interest. In the domain of human collaboration that monitored object may be another actor, a group of actors and their interactions, or a certain task or activity. For example, a boss may monitor a subordinate and a group leader may monitor other group members and their interactions.

2.2 Subtypes of Monitor Pattern

Besides the different types of observed objects listed in the previous subsection, Monitors can be classified by the motivation underlying their action and by the methods they have at hand to monitor their object. We differentiate between four types of monitoring based on the motivation underlying the monitoring activity.

Informational Monitoring: The actor monitors an object he is not directly related to or affiliated with. His motivation for monitoring is not obvious to others. An example is a team member monitoring the activities of a more experienced colleague in order to learn from him.

Observational Monitoring: A dependency between the actor and the object he monitors is given. This dependency is the motivation for monitoring since he may be influenced by or may have to react to certain events related to the monitored object. An example is a team member monitoring a task which his own task depends on, for example, his own task relies on input generated by the other task.

Supervisional Monitoring: The Monitor is responsible for or has authority over the object he monitors. He has the right (or the obligation) to intervene when necessary. An example drawn from another pattern is a master monitoring the activity of his slaves to ensure correct execution of the (sub) tasks he assigned to them.

Coordinational Monitoring: The Monitor is responsible for coordination of a team (or parts thereof) and for efficient allocation of resources. He is not interested in the details of an activity, but only in information regarding an object's status, for example, availability of a person, or progress of a task.

2.2.1 Student, or Studying Monitor

Definition: A Studying Monitor has no direct relationship or dependency to the monitored object and his monitoring activity is not driven by an external necessity.

Characteristics: Studying Monitors may remain undetected by the object that is being monitored. Typically, a Studying Monitor does not directly interact with the corresponding object, nor does he request active reporting from it.

Methods of Monitoring: Since the Studying Monitor is not affiliated with what he monitors his methods of monitoring are limited. Typically, all monitoring activities will have to be initiated by himself and the information he may acquire may be limited as well. Monitoring must not be intrusive, and any direct access to the monitored object is granted on a purely "voluntary" basis.

2.2.2 Dependant, or Dependent Monitor

Definition: A Dependent Monitor (DM) observes objects that have an impact on himself or the tasks he is involved in.

Characteristics: The Dependant Monitor is best characterized by his ability to interact with his monitored object while not having any authority over it and thereby not being able to interfere with the object's activities.

Methods of Monitoring: Due to the explicit dependency of the Monitor and the object of interest, Dependent Monitors are granted certain rights and opportunities to guarantee them sufficient information. For example, Dependent Monitors may be allowed not only to monitor publicly available information, but may also be given the opportunity to directly address, for example, actors involved in the monitored task, when further information is needed.

If provided by the object, DMs may be permitted to subscribe for notifications that a given object issues to dependent entities. In case of critical dependencies, a DM may even have the right to demand such notifications.

2.2.3 Supervisor, or Supervising Monitor

Description: A Supervisor is responsible for or has the authority over the object he monitors. He also has the right to manipulate the object, for example, issue orders to a person, or influence the execution of a task.

Characteristics: The Supervisor's responsibility for a given object implies both the right and the duty for active intervention whenever necessary. A Supervisor is the only subtype of a Monitor with the right and opportunity to directly manipulate the object.

Also, a Supervisor may have to be available to objects he monitors when they request guidance. Therefore, he may have to provide support upon request.

Methods of Monitoring: Along with a Supervisor's control over the monitored object comes the right to enforce active reporting by the object of interest in whichever way considered to be suitable by the Supervisor. This may include notifications regarding relevant events, status or progress reports in desired intervals, or even personal reports to him by a monitored actor.

2.2.4 Coordinator, or Coordinating Monitor

Description: A Coordinator monitors the status of objects in order to coordinate activities and to enable efficient allocation of resources.

Characteristics: A Coordinator is not interested in the actual "content" of activities, e.g., the content of documents resulting from collaborative activities, but only in the status of objects. Significant status information could be availability of actors, e.g., "available" or "unavailable for 2 more hours", and progress of tasks, e.g., "90% completed", or "estimated completion in 3 days".

Methods of Monitoring: Considering the importance of effective and efficient coordination in human collaboration, a Coordinator should typically not be denied any information he considers to be relevant. Therefore, Coordinators would be granted rights regarding information they have access to. Also, for example, in time critical situations the monitored object may do more frequent reporting to the Coordinator.

3 MESSAGE ANALYSIS

As we stated in the introduction, today's most widely used messaging protocols provide very little support to the user in terms of message addressing, message management, and message prioritization.

In the following discussion we present a simple, yet effective way of annotating messages with machine readable information to facilitate computer-supported collaboration using basic messaging technologies. The tags which are embedded in messages are specified in XML and can therefore even be considered to be human-readable, even though this factor did not play an important role in the design.

The information contained in these tags may be used at two stages:

- *Ongoing collaborations*: Applications include semi-automatic message addressing and message management, for example, ordering of messages by activity, archiving

of messages relating to completed tasks, and message prioritization.

- *Post-collaboration*: Tags allow for message archiving, and improve message analysis opportunities, possibly as far as extracting patterns and workflow information.

Annotations should allow the mapping of messages to an activity context, which in turn may relate to a task, a project, and thereby a team. Consider a hierarchically organized project as depicted in Figure 2.

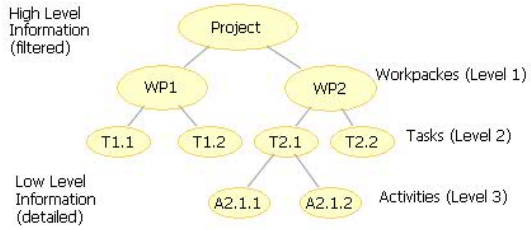

Figure 2: Granular levels of detail in project topology.

As displayed by Figure 2, the root of the hierarchical tree is the Project. It provides the container for logical workpackages and furthermore actual tasks which are being executed by team members to contribute to collaborative activities. Workpackages and tasks include properties such as start time, end time, deadlines and milestones, expected duration, outcome (e.g., artifacts), and resources. To work effectively in teams, coordination of team members and progress tracking is required. Measures at different levels of detail optimize collaboration and minimize risks that may arise. For example, at level 2 in Figure 2 we measure task progress, deviations, and possibly identify deadlocks.

Our work in the area of message based systems aims at structuring ad-hoc collaboration by applying a flexible topology to manage information, which allows users to create projects, to associate a number of tasks to projects, to define a responsible person or leader for a particular task, to assign members or contributors to tasks, and to associate a set of artifacts to tasks.

Exemplary messages relating to information management, including XML notation, are “create project” `<create-project />`, “create task“ `<create-task />`, “update status“ `<update-status><task-id /></update-status>`, “request approval“ `<request-approval><task-id /><artifact-id /></request-approval>`. Read-only requests include, for example, “query status“ `<query-status><task-id /></query-status>`.

In the following subsections we elaborate on collaboration and communication (or the protocol) patterns and define an XML based method to coordinate collaboration in ad-hoc teams that communicate in peer-to-peer mode.

3.1 Communication Patterns

A communication pattern describes the structured and periodic exchange of messages. The sequence diagram in Figure 3 shows a simple “create task” pattern.

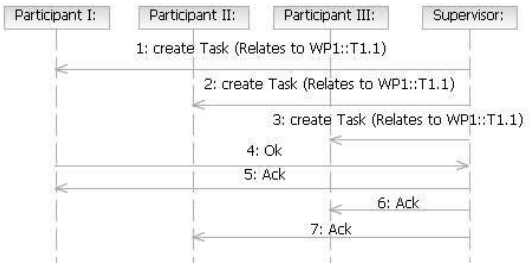

Figure 3: Create new task.

The Supervisor assigns a new work item to Participant I, II, and III by sending a “create task” message (messages, in short **m**, 1-3). Figure 3 depicts the case where Participant I holds the task leader role. Therefore, Participant I accepts and confirms the assignment by responding with “Ok”. Finally, the Supervisor acknowledges that “create task” was successful (“Ack” to each Participant, **m** 5-7). As a result of this interaction, a new task (i.e., task named T1.1 with logical association to workpackages WP1) is created, assigned, and saved to the persistent store (e.g., XML data saved in the local file system – no central database server is required).

A task typically produces some output in the form of artifacts such as documents or reports. These artifacts are associated to or referenced by a particular task. In our example the task leader, Participant I, coordinates the work among contributors (Participant II and III). A final approval or review, however, may be required by the Supervisor. Figure 4 illustrates an approval pattern. Participant I sends a “request approval” message, containing the “relates to” element that refers to specific activities or tasks, in our example T1.1 scoped by WP1, to the Supervisor (**m** 1). The

"relates to" element helps us to inspect the context of the message and to associate messages to activities and tasks. Additionally, we can determine the causal dependencies between messages and organize them in a structured way. For example, messages relating to a specific task or workpackage can be represented in form of a message tree.

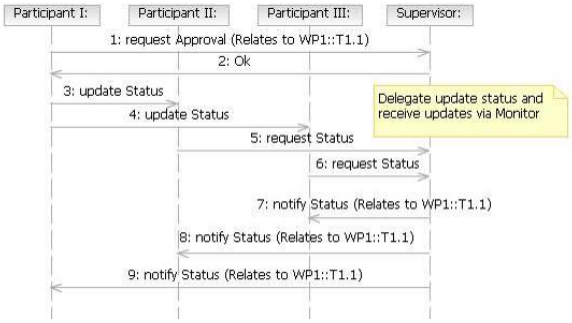

Figure 4: Request approval from Monitor.

The Supervisor approves T1.1, thereby authorizing associated or relevant artifacts, by returning "Ok" (**m** 2). Please note, the semantic meaning of "Ok" in this context is to confirm the request – "request approval", which may in fact contain an "accept" or "reject". For simplicity we abbreviated the approval message as "Ok", however, it should be seen as "Ok (Relates to WP1::T1.1)".

In Figure 4, we assume that the approval was only sent to Participant I as a "private" message. Upon receiving the approval, Participant I distributes an "update status" message (**m** 3-4), which can be regarded as a **command** or **delegate** to receive updates. In other words, it can be seen as an invalidate-status command (by sending "update status"), which results in "request status" messages to be sent (**m** 5-6).

The approval pattern in our example is denoted by following properties:

- The "request approval" message is sent by the leading participant (task leader).
- Approval is given by the supervising Monitor.
- Final status information (e.g., status of WP1::T1.1) has to be obtained from an authorized entity, in our case the Supervisor.

Following the approval pattern, Participant II and III query the Supervisor for updated status information (**m** 5-6). The Supervisor, in turn, notifies all Participants regarding T1.1 updates (**m** 7-9).

4 INTERACTIONS THROUGH EMAIL

Our pattern based approach to coordination problems in collaboration, specifically in ad-hoc collaboration, by means of message annotations in form of embedded XML tags, can be applied to any message oriented system that has basic features such as addressing, a text based protocol, and the ability to store messages. Instant Messaging (IM) and Email are two prominent examples for such systems and are widely used in collaboration. Email is the most extensively used technology in asynchronous (ad-hoc) collaboration because of its flexibility and its ability to interoperate with any other email client/program across organizational boundaries and company firewalls.

However, flexibility comes at a price. There is a tradeoff between versatile collaboration and structured or even rigid collaboration flows (Dustdar, 2004). We validate our proposed concepts by applying them to email-based collaboration. In the next sections we provide a high-level specification of our message annotation framework.

4.1 Message Annotations

Create New Project: Users have the ability to create a new project, in order to initiate a new collaboration, by providing information such as project name, description, start, end, resources, etc.

User can make this information available (i.e., announcement that a new project exists) by sending a message to a predefined distribution list, similar to a multicast "invite" message. Whoever is interested from that list in joining the project creates a response which relates to the announced project. The other option is to specify members explicitly by selecting them from a list.

A fragment of the corresponding XML representation (XML tag) that is embedded (along with exemplary data) in an email message:

```
<create-project>
   <id>project123</id>
   <name>Class data mining</name>
   <begin /><end />
   <team>
```

```
    <member><name /><email />
        <role />
    </member>
  </team
</create-project>
```

All members receive the message. A client application discovers the machine processible information, i.e., XML tag `<create-project>`, in the message and prompts for information regarding the project and whether or not the user wishes to participate. If the user confirms, a message is being generated and automatically sent to all other members. An exemplary confirmation message:

```
<join-project>
  <project>
  <id>project123</id>
  </project>
  <member><name /><email /><role />
  </member>
<join-project>
```

A new folder can be created automatically in the users email message repository (logical or physical) to structure all project related messages and activities. Upon receiving the incoming message regarding the addition of a new team member (message holding tag information `<join project>`), project related information that is saved on the other participants' clients in a local store is being updated automatically and reflected in the user interfaces by means of event notification. From this point on, the project is saved in a persistent store and members can associate messages to the project. Every member has the ability to a) create tasks, b) update status information, c) request status information, d) send information and artifacts to members (however, as opposed to standard email, in a structured way where embedded artifacts, resources, etc. correspond to tasks and to an *activity context*), and furthermore, which is not further elaborated in this paper, e) initiate and schedule meetings, and f) request group decisions (distributed decision making support).

Create Task: A new task is created by providing the user with the ability to choose a project from the local store. Once a project is selected, the user selects the recipients of the task and may assign predefined roles to members such as a) Leader/Owner, b) Member, or c) Monitor.

A message is sent to the respective participants, holding different roles (Leader and Members) and a carbon copy (CC) to Monitors, containing:

```
<create-task>
  <name /><desc /><duedate />
  <leader /><members /><monitors />
</create-task>
```

Upon receiving this message, users need to confirm (e.g., "Do you want to add task T to this project and accept your role?"). A logical folder for messages relating to that task is created under the corresponding project folder.

Update Task Status: Task members can send messages regarding task progress using "update task status" messages. The corresponding XML annotation for "update status" along with (implementation dependent) possible values:

```
<update-status><task-id />
  <progress>50%</progress> (or "on
  time", "delayed", "completed")
</update-status>
```

As task status updates are received, the persistent store is updated on each participant's client.

Request Task Status: The team can exchange and request task status information messages. This feature is provided by "query status" that lets users select from projects or teams and depending on project selection pick specific tasks. As we show at a later point in more detail, the message may be distributed based on roles and collaboration patterns, for example, by requesting status information from other peers or by requesting status information only from authorized entities such as Monitors (e.g., Supervisors). An exemplary "query status" message:

```
<query-status>
  <task-id />
  <last-update>TimeStamp</last-
  update>
  <confidence>60%</confidence>(or
  "low", "medium", "high")
</query-status>
```

Approval: Our framework provides the ability to issue a "request approval", as illustrated in Figure 4, which relates to a task or an associated artifact. A message is sent to the person who should give the approval containing:

```
<request-approval>
  <task-id />
  <task-leader />
  <progress />(e.g., "completed")
  <artifact-id />
```

```
</request-approval>
```

The corresponding response message would be:

```
<approval>
   <task-id />
   <approved /> (e.g., accept, reject,
   pending, deferred)
   <approved-by />
   <reason />
</approval>
```

An approver may accept or reject a task or the actual outcome of a task in form of artifacts or set the approval status to "pending". Furthermore, it is possible that the approver delegates the decision to another entity or Monitor. In this case, a custom XML tag with an element `<relates-to>`, that refers to the delegated approver, may be generated.

Retrieve Project Status: The ability to retrieve a project's status information is a vital part of our system. Previous elements such as "create task" or "update task" enable users to manage information and messages in a structured way – suitable for coordinated collaboration (e.g., see "request approval" use case). *Retrieve project status* aims at providing a high level overview of activities and tasks that are going on in a particular project (e.g., as illustrated in the project tree in Figure 2). It allows the users to get summaries of the overall status (e.g., management summaries) – at a glance – and also allows new team members to quickly get started in an ongoing project (bringing new team members up to speed by generating custom reports and project summaries). In particular, it allows the generation of a report of the current project status, including team members, tasks and their status. Summaries can be displayed as simple reports including history of contributions such as file changes, and also reports suitable for new team members by getting all relevant information that other members have stored on their systems. This could even include the most recent version of all artifacts such as files related to the project. An exemplary XML report could comprise the following elements:

```
<project-status>
   <project-info /> (e.g., id, name,
   workpackages, etc.)
   <team-info /> (e.g., active
   members, monitors)
   <task-info /> (e.g., not yet
   started, completed, pending)
<project status>
```

4.2 Collaboration Use Case

A slightly more complex interaction scenario is depicted by Figure 5. Participant I updates status information, related to T1.1 (**m** 1-2), which in turn triggers an "update status" initiated by Participant II (**m** 3). In this case Participant II is monitored by a Person-Dependent Monitor. In principle, we distinguish between monitoring a person and monitoring a task/activity. Next, we see a protocol specific pattern that relies on roles or even specific assignments in the form of tasks. Participant I sends an "update status" message to his/her Supervisor. A *handshake* mechanism in form of "ACK" and "OK" messages is applied to guarantee delivery of status information (**m** 4-6).

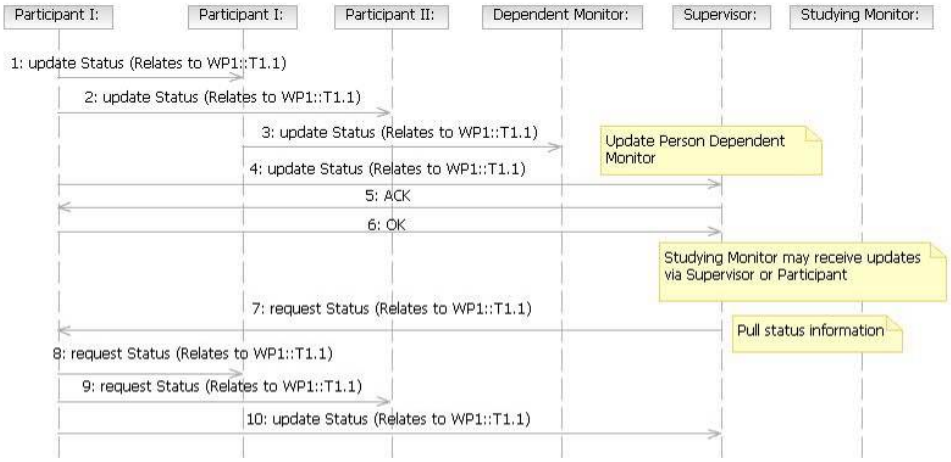

Figure 5: Updating task information interacting with multiple Monitors.

As a next sequence in Figure 5 we see "request status" information. The supervisor pulls status information regarding T1.1 from the task leader (**m** 7), i.e., Participant I, asynchronously. In turn, the leader requests status information from each participant (**m** 8-9) to ensure consistency of status information and provides consolidated information in form of an "update status" message to the Supervisor (**m** 10).

5 RELATED WORK

In our previous work we have introduced patterns from the software engineering domain, i.e., Proxy, Broker, and Master/Slave (Gamma et al., 1994), as a metaphor for human collaboration patterns. (Dustdar and Hoffman, 2006). These patterns can be utilized to make collaboration more efficient and also to establish team awareness. (Gombotz et al., 2006). As the number of messages sent in collaboration grows, it becomes increasingly challenging to process them. Additional socially salient information may be needed to bring important emails to the user's attention. (Neustaedter et al., 2005), (Petrie, 2006). Data obtained from field studies suggest that email activities may be categorized in: flow, triage, task management, archive, and retrieve. (Venolia et al., 2001). Email archives and traces of communication and coordination activities can be utilized to perform post-collaboration analysis and extract relations in human collaboration. Social networks can be used to visualize these relations and dependencies in a graph representation. (van der Aalst et al., 2005).

6 CONCLUSIONS

We presented an extensible XML based framework that allows users to exchange collaborative messages and information in a structured way. Annotations in messages can be used to organize messages (semi-) automatically based on activity contexts. Reports and summaries can be generated automatically in order to understand the high level status of a project or to assist team members that are joining the team or have been absent for some time to better understand past activities and current status. The presented XML tags, which are embedded in messages, can be used for post processing and message analysis to identify and extract patterns and possibly workflow information.

Our pattern based collaboration framework is fully distributed and does not rely on any central server. However, if teams become large and collaboration lasts for a long period of time, a server that saves XML annotations and coordinates activities based on patterns may be employed. Although presented in the context of email, methods and principles of our framework may be applied to any messaging-based system.

ACKNOWLEDGEMENTS

Part of this work was supported by the EU STREP Project inContext (FP6-034718).

REFERENCES

van der Aalst, W.M.P., Reijers, H. A., Song, M., 2005. Discovering Social Networks from Event Logs. *Computer Supported Cooperative Work* 14(6), 549-593.

Dustdar, S., 2004. Caramba - A Process-Aware Collaboration System Supporting Ad Hoc and Collaborative Processes in Virtual Teams. *Distributed and Parallel Databases*, 15(1), 45-66. Kluwer Academic Publishers.

Dustdar, S., Hoffmann, T., van der Aalst, W.M.P., 2005. Mining of ad-hoc business processes with TeamLog. *Data and Knowledge Engineering*, 55(2), 129-158, Elsevier.

Dustdar, S., Hoffmann, T., 2006. Interaction Pattern Detection in Process Oriented Information Systems. *Data and Knowledge Engineering*. Elsevier.

Gamma, E., Helm, R., Johnson, R., Vlissides, J., 1994. *Design Patterns – Elements of Reusable Object-Oriented Software*. Addison-Wesley.

Gombotz, R., Schall, D., Dorn, C., Dustdar, S., 2006. Relevance-Based Context Sharing through Interaction Patterns. *The 2nd International Conference on Collaborative Computing*. IEEE Press.

Neustaedter C., Bernheim Brush A.J., Smith, M. A., 2005. Beyond "From" and "Received": Exploring the Dynamics of Email Triage. *CHI 2005*.

Petrie, C., 2006. Semantic Attention Management. *IEEE Internet Computing*, 10(5), 93-96, Sept/Oct, 2006.

Schümmer, T., 2005. A Pattern Approach for End-User Centered Development Groupware Development. *EUL Verlag*.

Venolia, G.D., Dabbish, L., Cadiz JJ, Gupta A., 2001. Supporting Email Workflow. *MSR-TR-2001-88*.