Integration of collaboration and interaction analysis mechanisms in a concern-based architecture for groupware systems

Rafael Duque\textsuperscript{a,∗}, María Luisa Rodríguez\textsuperscript{b}, María Visitación Hurtado\textsuperscript{b}, Crescencio Bravo\textsuperscript{a}, Carlos Rodríguez-Domínguez\textsuperscript{b}

\textsuperscript{a} Departamento de Tecnologías y Sistemas de Información, Universidad de Castilla-La Mancha, E.S.I., Paseo de la Universidad 4, 13071 Ciudad Real, Spain
\textsuperscript{b} Departamento de Lenguajes y Sistemas Informáticos, Universidad de Granada, E.T.S.I.I.T., c/ Periodista Daniel Saucedo Aranda s/n, 18071 Granada, Spain

A R T I C L E I N F O

Article history:
Received 5 March 2009
Received in revised form 4 May 2010
Accepted 15 May 2010
Available online 31 May 2010

Keywords:
Collaboration and interaction analysis
CSCW
Groupware
Software architectures

A B S T R A C T

Collaboration and interaction analysis allows for the characterization and study of the collaborative work performed by the users of a groupware system. The results of the analyzed processes allow problems in users’ collaborative work and shortcomings in the functionalities of the groupware system to be identified. Therefore, automating collaboration and interaction analysis enables users’ work to be assessed and groupware system support and behavior to be improved. This article proposes a concern-based architecture to be used by groupware developers as a guide to the integration of analysis subsystems into groupware systems. This architecture was followed to design the COLLECE groupware system, which supports collaborative programming practices and integrates an analysis subsystem that assesses different aspects of the work carried out by the programmers and adapts the functionality of the system under specific conditions.

© 2010 Elsevier B.V. All rights reserved.

1. Introduction

In recent years there has been a growing demand for technologies that enable collaboration among users involved in a common task \cite{34}. In an attempt to adapt to this scenario, software systems have evolved from single-user approaches towards the use of shared workspaces aimed at supporting the collaborative work of groups of users. These software systems, known as groupware systems, are currently equipped with multiple mechanisms for supporting interaction among the members of a group or the manipulation of objects in shared workspaces. The requirements needed for a successful groupware system are being studied by the CSCW (Computer-Supported Cooperative Work) research field, and range from designing and developing suitable technological solutions to considering the social aspects of an organization \cite{22}.

At present, one of the main challenges CSCW is facing involves the development of computational methods and models for automating processes of collaboration and interaction analysis. These analysis processes focus on capturing and modeling the interactions between users in shared workspaces in order to characterize the collaborative work of a group. There is a wide range of motivations behind the development of analysis support, such as web-based systems that require mechanisms for evaluating how the system is used \cite{3}, information systems using models for the study of the collaborative work through mining techniques \cite{18}, information systems collecting user histories in order to improve knowledge management \cite{32}, or systems that support and measure collaborative software engineering practices \cite{8}.

Although collaboration and interaction analysis arose from the need to represent and understand users’ activity in a collaborative environment, it is not only the CSCW area that has undertaken such analysis tasks; many research fields deal...
with this type of analysis from different perspectives. In knowledge management, social computing models are used to analyze each user action according to its interest for the users of a community [23]. Business Process Analysis studies the user’s actions to assess their impact in terms of achieving business goals [49]. Computer-Supported Collaborative Learning (CSCL) deals with the analysis of learners’ collaboration based on pedagogical theories and the evaluation of learners’ behavior and knowledge acquisition [45]. In all cases, the analysis processes are based on the definition of models for specifying how to process (capture and transform) the information stored in the log documents produced by the groupware system. The log documents usually contain a trace of users’ work within shared workspaces. However, none of these proposals tackle the integration of mechanisms for automating the analysis process into groupware systems according to the specific goals of each case.

In order to deal with the problems mentioned above, this article proposes a concern-based architecture aimed at enabling developers to incorporate collaboration and interaction analysis mechanisms into groupware systems. Groupware systems usually support actions that enable communication, coordination and collaboration between the users involved in a common task [16]. The underlying architectural model includes a set of components that interact amongst themselves in order to capture and characterize the users’ actions and to suitably apply this characterization in order to improve performance (e.g. visualizing analysis results, adapting the behavior of the groupware or its user interface, etc.).

The rest of the article is arranged in six additional sections. Section 2 presents the purpose and main phases of the collaboration and interaction analysis and the main architectural models used for groupware systems design. Section 3 describes a concern-based architecture that enables the development of groupware systems that integrate analysis facilities. Section 4 describes an experience in which the architecture was followed to design the COLLECE (COLLaborative Edition, Compilation and Execution) groupware system, and in particular its analysis subsystem. Section 5 makes an evaluation of the proposed architecture. Section 6 reviews the methods and techniques used for collaboration and interaction analysis in groupware systems, which are compared to the work presented here. Lastly, Section 7 presents the conclusions drawn from this research and proposes a future line of work.

2. Research context

2.1. Collaboration and interaction analysis

Collaboration and interaction analysis is an instrument for representing and studying users’ work within a groupware system. This analysis task, which presents clear benefits with regards to understanding users’ behavior and evaluating the use of the system, also contributes useful information and methods for providing awareness information that enables members of a workgroup to be aware of the shared objects, evolution of the shared workspace and cooperative activities in groupware systems [4]. For instance the collaboration and interaction analysis processes are the starting point for providing users with perceptible improvements to the development of collaborative activities through visualizations of evaluations of the completed work or groupware system adaptations. Most collaboration and interaction analysis systems propose an analysis life cycle made up of three phases [6]:

- **Observation**: This captures and collects the information needed to subsequently infer analysis indicators, i.e. significant variables that evaluate specific aspects of the collaborative work process. These variables can be quantitative, when they are presented in numeric form, or qualitative, when they provide textual information used to evaluate certain aspects of the collaboration.
- **Abstraction**: This uses the information stored in the observation phase to compute higher-level analysis indicators. Artificial intelligence techniques and mathematical functions are usually used in this inference process.
- **Intervention**: This uses the inferred analysis indicators to improve the collaborative work process (e.g. by adapting the groupware user interface, proposing new forms of collaboration, giving advice to the users, etc.).

The three analysis phases are specified by users who play the role analyst. The analyst can be an external evaluator who is interested in assessing different aspects of the collaborative work (productivity, coordination, communication, etc.) with a definite purpose in mind or, for example, a teacher who wants to evaluate the work carried out by a group of students within a CSCL environment. An analyst is responsible for defining the kind of information (typically raw data) to be considered in the observation phase and how it should be stored. Moreover, the analyst must specify the analysis indicators to be inferred in the abstraction phase and the steps to be followed in the intervention phase in order to improve the collaborative work. The analyst must warn the groupware users that a dataset about their activities will be captured during the analysis and this dataset will be used in accordance with ethical and legal rules.

This article focuses on the proposal of a concern-based architecture to allow developers to integrate analysis subsystems with the aforementioned characteristics into groupware systems. Such analysis subsystems include mechanisms and functionalities for automating the observation–abstraction–intervention analysis life cycle.

2.2. Groupware architectures

Groupware systems are complex to construct, involving difficult technological problems such as the need to support real-time interaction across distributed systems [52]. This intrinsic complexity of groupware systems demands great effort
with regards to specifications and development [2]. Therefore, one important question involves how to define architectural models that allow designers to specify and create groupware systems from a high abstraction level rather than starting immediately with the implementation [43]. The architecture of a software system can be defined as “the fundamental organization of a system, embodied in its components, their relationships with each other and with the environment, and the principles governing the design and evolution of this system” [28].

Phillips [40] presents some of the more widely recognized groupware architectures and classifies them as follows:

- Reference models, which specify the complete structure of some types of systems at a relatively large granularity. The reference model enables the groupware system to be easily mapped onto it.
- Architecture styles, which present a solution to a particular problem as a vocabulary of components, connector types and their permitted interaction patterns.
- Distribution architectures, which represent the distribution of the system functionality across connected computing platforms.

Dewan’s reference model [9] describes a generic architecture that encapsulates architectural properties common to a wide range of collaborative systems and identifies a set of issues (single-user architecture, concurrency, distribution, collaboration awareness and versioning/replication) that an architecture must deal with.

Clover’s architecture model [33] for groupware systems is a result of the combination of the layered approach of Dewan's generic architecture with the functional decomposition of Clover’s design model. This design model defines the classes of services that a groupware application may support, namely, production (the objects produced by a group activity or the objects shared by multiple users), communication (person-to-person communication) and coordination (temporal relationships between multi-user activities and relationships between actors and activities) services.

The Workspace model [41], a multi-level architectural style for synchronous groupware, provides a clear separation of the conceptual structure from the distributed implementation. The model includes a formally-defined, distribution-transparent, conceptual level architectural model with appropriate abstractions for the development of groupware, and a formally-defined, implementation level architectural model that sets out the distributed system issues abstracted at the conceptual level.

Finally, the Jazz Integration Architecture (JIA) [29] is a reference model that enables the connection and integration of sets of software tools. JIA includes a central server that offers a group of web services. Some of these services allow the users of the interconnected tools to work collaboratively [7].

However, none of these groupware architectures consider components that support functionalities of collaboration and interaction analysis. This gap has forced developers to design ad hoc analysis mechanisms in each case. The next section fills the aforementioned gap with the description of a concern-based architecture intended to guide developers in the design and integration of collaboration and interaction analysis mechanisms into groupware systems.

3. A concern-based architecture for groupware applications incorporating analysis support

Traditional approaches to architecture design suffer from crosscutting features and behaviors, which are scattered and become tangled with the components and connectors, resulting in final architecture solutions that have difficulty evolving and are not easy to reuse. Our reason to propose a concern-based architecture for groupware applications is to enable separation of the artifacts that give rise to such concerns and to address the specific issues involved in collaboration and interaction analysis. In order to improve maintainability, reuse and evolve, it is necessary to detect the relationships between these artifacts.

The ISO/IEC 42010 standard [28] defines concerns as “those interests that pertain to the system’s development, its operation or any other aspects that are crucial or otherwise important to one or more stakeholders”. According to [46], the main advantages of applying separation of concerns in software architectures are: (i) decreasing the complexity of software development by concentrating on different issues separately, (ii) supporting the division of efforts and separation of responsibilities, and (iii) improving the modularity of software artifacts. Software artifacts thus tend to be cohesive loosely coupled and reusable.

In this section, we introduce a logical dimension of an architecture based on the separation of concerns and different abstraction levels. Each concern allows the common abstractions to be captured, and the logical dimension describes the logical structure of the groupware system in terms of the subsystems and their relationships [26]. Additionally, different concerns can be incrementally and modularly integrated into the system as new requirements (corresponding to groupware concerns) arise [48,46].

For our purposes, we extended an existing architecture. As such, the starting point for our approach is the AMENITIES methodology [19], which enables the design of groupware systems to be addressed systematically thus facilitating subsequent software development. This allows a conceptual model of a cooperative system to be built and focuses on the group concept. It also covers significant concerns of both group behavior (dynamism, evolution, etc.) and structure (organization, laws, etc.). The resulting specification contains relevant information: cooperative tasks, domain elements, person-computer and person-person interaction, etc. However, it does not include the description of specific analysis mechanisms for capturing and studying user actions, i.e. the collaboration and interaction between users.
3.1. Logical view

According to [20], the logical dimension of a groupware system should consider four subsystems: Identification, Meta-information, Awareness and the Application itself (Fig. 1). The proposed logical dimension adds, at a high abstraction level, an additional Analysis subsystem aimed at managing collaboration and interaction analysis. At a lower abstraction level, this subsystem comprises three subsystems, Observation, Abstraction and Intervention (see Fig. 1), which correspond to the analysis phases according to Bravo et al. [6].

The Identification (user’s access control), Meta-information (meta-data management) and Awareness (contextual information) subsystems are, in one way or another, always present in each groupware application. The Analysis subsystem is only present, and therefore developed in particular groupware systems, when collaboration and interaction analysis is considered to be a requirement. In order for a groupware application to integrate an Analysis subsystem, a user playing the analyst role must define the activities and tasks to be analyzed. The Analysis subsystem then accesses the information provided by the Awareness and Meta-information subsystems in order to observe and capture users’ activity. Although there is only one Analysis or Application subsystem in this example, additional instances can be easily integrated whilst still maintaining this same architectural design.

3.2. Functional view

As previously indicated, the logical dimension provides the relationships between subsystems. Since groupware applications are necessarily distributed, it is important to obtain an implementation from a set of subsystems that communicate with each other using well-defined interfaces. Fig. 2 shows the five subsystems and their use relations according to the functionality they provide. Each subsystem provides an interface designed to achieve independence between the subsystems and other applications that are using them.

The Identification subsystem provides the functionality needed to verify that a user has permission to access the groupware system. Its login module includes an authentication method that takes as input the user name and password, and then exchanges data with the Meta-information subsystem (see Fig. 2) to verify that the information provided corresponds to a registered user. In particular, the Identification subsystem is used to control user access to the system and to start the Application subsystem.

The Meta-information subsystem supports all of the functionality for managing meta-data (new roles acquired, tasks for each role, etc.) specified in the cooperative model derived from the application of the AMENITIES methodology. This subsystem provides the services to manage and configure all of the work sessions. A work session is defined by a set of groupware users who must complete a task. The Meta-information subsystem includes a sessions configuration manager module (see Fig. 2) that enables the information about the permissions of users of the groupware system, the floor control policies for changing the configuration of the shared workspaces and the tasks that the groupware users must complete to be managed.

The Awareness subsystem includes modules that process the information sent by the Application subsystem that characterizes the users’ work in the shared workspaces. This information is used by the Awareness subsystem to facilitate the users’ awareness of their colleagues and the activity. The Awareness subsystem includes the following three modules (see Fig. 2): (i) Shared state synchronization manager that enables information on the state of the collaborative process to
be displayed, (ii) **Tele-pointers synchronization manager** that provides information about each user’s tele-pointers and (iii) **Message synchronization manager** that provides the message distribution mechanisms that enable the synchronization of shared workspaces.

The **Application** subsystem supports functionalities that manage the interactions between the users and the groupware system. The **Application** subsystem includes the following three main modules (see Fig. 2): (i) the **Collaborative tools controller** that supports the interactions between the groupware user and the shared workspaces, (ii) the **Individual tools controller** that manages the individual workspaces, and (iii) the **Contextual panels controller** that manages those areas of the user interface displaying awareness information.

The **Analysis** subsystem provides all of the necessary services for collaboration and interaction analysis. This subsystem is made up of three subsystems (see Fig. 2). The **Observation** subsystem provides a set of functionalities that allow for the collection and recording of the actions performed by the users. Only those actions that have been selected by the **analyst** are processed. This is known as filtering. As such, the **Filtering provider** provides the **analyst** with support for designing models to specify the actions which are to be captured by the **Observation** subsystem. The **Event storage** stores the events detected by the **Event provider**, which captures any kind of event. Lastly the **Filtering manager** confirms that one of these events matches an action defined in the filtering model that has been specified by the **analyst**.

The **Abstraction** subsystem processes the information stored by the **Observation** subsystem to automatically infer statistical information and variables (analysis indicators) for characterizing the collaborative work. The **Inference engine** module includes mathematical functions and inference rules (e.g. based on bayesian networks or fuzzy logic) to compute analysis indicators. The methods included in the **Analysis indicators manager** collect and store the completed analysis indicators so that they can be accessed and manipulated later.

Finally, the **Intervention** subsystem invokes methods of the **Analysis indicators manager** to obtain the analysis indicators that are needed to intervene. This information enables the **Visualizer** module to show analysis indicators and advice. The **Intervention** subsystem also uses the **Intervention manager** module to interact with the **Meta-information**, **Awareness** and **Application** subsystems to adapt the workspaces configuration. The **analyst** designs the interventions that should be performed by the **Intervention manager**, which refers specifically to changes to the form or behavior of the workspaces as well as the system itself. This is of course subject to the functionality that is available in the groupware system.
is a tailoring by customization approach [44] in which the Intervention manager selects from among a set of pre-defined configuration options in the groupware system. Some non-exhaustive examples of this are to (i) change the order in which the different tasks are put to the users, (ii) change the policies of coordination in the floor control tools, or (iii) alter the user interface in some way. The interventions of the adaptation type are therefore limited to the functionality or modes of operation already implemented by the groupware. An example might be a groupware system that supports a wide variety of coordination and floor control policies, and an intervention mechanism that would choose the most suitable policy in each situation according to the inferred analysis indicators. However, in cases where the analysis suggests an intervention which is not supported by the functionalities available on the groupware system, a software maintenance process would need to be carried out in order to incorporate the new functionalities required into the system.

A software product family, in which the software systems share common features, can be derived from the proposed architecture. A product family is a set of systems built from a common set of software assets [1]. The software systems of a product family include commonalities and variabilities [21]. The commonalities are general features included in all of the software systems of the product family. The variabilities are specific features that focus on an individual system. On the one hand, each of the software systems derived from the proposed architecture presents some basic commonalities: they follow the same observation-abstraction-interaction analysis life cycle, and they are based on an architectural pattern made up of the Identification, Meta-information, Awareness and Application subsystems. The three phases of analysis also contain commonalities, such as the use of models for filtering user or group actions, the analysis indicator concept, the use of mathematical functions and inference rules for calculating indicators, and the use of intervention mechanisms for improving the collaborative work. On the other hand, the features that can be considered variabilities are those specific actions that are collected and stored in the observation phase, the particular analysis indicators considered for a specific collaborative process, the specific inference schemes for calculating high-level indicators, the particular interventions to be performed, and so on.

In some cases a concern cannot be related to a specific subsystem, since it is a transversal concern that affects an entire set of subsystems. In addition to the concerns presented above, the following three main transversal concerns are considered in our architecture, two of which are inspired by the classical time–space taxonomy of groupware [30]:

- **Time-related interaction type (synchronous/asynchronous):** The main characteristic of a groupware system is its ability to notify the rest of the members of its/her workgroup about the activity of a user (e.g., changes in the data). In synchronous groupware systems, these notifications take place instantaneously, whereas in asynchronous systems, timing is not as important and the users do not need to be notified in real time, since the nature of the supported tasks normally suggests individual and reflective work and the manipulated artifacts are only updated when the users request it, which conditions the participation of the Analysis subsystem. As such, in synchronous collaboration the Analysis subsystem, and particularly its Intervention subsystem, is more active, collecting raw data, inferring indicators and intervening usually in real time, considering the work of the whole group in a natural way. However, in an asynchronous setting the Analysis subsystem is usually more passive, focusing on the work of one user and only analyzing on demand, since it is not expected that partners are to be working on the same task at the same time.

- **Place-related interaction type:** When groupware users collaborate in a distributed way, all of the communication and coordination mechanisms are mediated through groupware. Therefore, the Observation subsystem has to collect all of the interactions that make up the collaborative work and this is analyzed in its entirety. However, where the users collaborate and are located in the same place (i.e., they share a workstation or interact on an interactive electronic whiteboard, etc.), they can establish face-to-face communication and coordination without the control of the groupware system. Therefore, in these cases the Observation subsystem cannot collect all the interactions and the Abstraction subsystem does not have the information needed to perform a complete analysis.

- **Task types:** The Meta-information subsystem manages the tasks to be carried out by the users. These tasks can be oriented towards solving a problem (i.e., create a model satisfying a set of requirements and constraints). In this case, the Analysis subsystem should automate the assessment of the degree to which the objectives set in the problem have been achieved. A second possibility is that the tasks do not consist specifically in solving a problem to fulfill some objectives, as is the case of systems that support conversations. In this case, the Analysis subsystem does not concern itself with analyzing the solution to the problem (e.g., the produced artifact).

### 3.2.1. Interfaces between subsystems

The Identification, Meta-information, Awareness and Analysis subsystems define interfaces between each other and with all of the possible groupware applications providing common services. The specification of these interfaces responds to the need not only to model the groupware system as a set of high-granularity components, but also to define the communication interface between subsystems. Table 1 shows an excerpt of the main services provided by each subsystem, specifically in relation to the analysis task.

Table 1 shows the names of the main services or operations included in the interfaces of each of the subsystems that are involved either directly or indirectly in the interaction and collaboration analysis. The Identification subsystem provides an operation that enables the user whose actions are going to be analyzed to be identified. The operations of the Meta-information subsystem provide information about the collaborative process to be analyzed. The Awareness subsystem
Table 1
Examples of operations provided by subsystems interfaces.

<table>
<thead>
<tr>
<th>Subsystem</th>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identification</td>
<td>Identify</td>
<td>Checks whether the identity provided by a user (login) is correct</td>
</tr>
<tr>
<td></td>
<td>Get_actors_list</td>
<td>Retrieves the list of groupware users (connected or disconnected)</td>
</tr>
<tr>
<td></td>
<td>Get_actors_connected_list</td>
<td>Retrieves the list of connected users</td>
</tr>
<tr>
<td></td>
<td>Get_tasks</td>
<td>Retrieves the list of tasks to be carried out by the users (i.e. create a model, produce a document, etc.)</td>
</tr>
<tr>
<td></td>
<td>Get_task_actions</td>
<td>Retrieves the list of actions relating to a task (e.g. an editing task contains actions such as insert, delete or modify text)</td>
</tr>
<tr>
<td></td>
<td>Get_workspace_policies</td>
<td>Retrieves the list of collaboration control policies of a workspace (e.g. access, floor control, etc.)</td>
</tr>
<tr>
<td></td>
<td>Get_tool_use_modes</td>
<td>Retrieves the list of modes for using a tool (e.g. with the chat tool, the users can either use only a set of predefined communication acts or they can use it freely)</td>
</tr>
<tr>
<td>Awareness</td>
<td>Notification_user_presence</td>
<td>Displays a message to inform that a user is logged in</td>
</tr>
<tr>
<td></td>
<td>Notification_user_message</td>
<td>Displays the same message to all the members of the group</td>
</tr>
<tr>
<td></td>
<td>Notification_all_users_message</td>
<td>Displays a message to a particular member of the group</td>
</tr>
<tr>
<td></td>
<td>Notification_task_state</td>
<td>Displays information about the task's state</td>
</tr>
<tr>
<td></td>
<td>Notification_action_state</td>
<td>Displays information about an action (type, time, workspace, user, etc.)</td>
</tr>
<tr>
<td></td>
<td>Notification_workspace_state</td>
<td>Displays information about the workspace's state</td>
</tr>
<tr>
<td></td>
<td>Notification_telepointer_state</td>
<td>Provides information about a tele-pointer (position, color, user, etc.)</td>
</tr>
<tr>
<td>Observation</td>
<td>Set_filter</td>
<td>Records an observation model to specify the kind of actions to be analyzed</td>
</tr>
<tr>
<td></td>
<td>Check_action</td>
<td>Verifies that an action should be collected and stored for analysis because it is included in the observation model that specifies the actions to be filtered</td>
</tr>
<tr>
<td></td>
<td>Store_action</td>
<td>Records an action that was previously captured and filtered</td>
</tr>
<tr>
<td></td>
<td>Select_user_interaction_actions</td>
<td>Retrieves the actions performed by a particular user</td>
</tr>
<tr>
<td></td>
<td>Select_group_interaction_actions</td>
<td>Retrieves the actions performed by the group of users</td>
</tr>
<tr>
<td>Abstraction</td>
<td>Insert_inference_rule</td>
<td>Includes a new inference rule for computing analysis indicators</td>
</tr>
<tr>
<td></td>
<td>Compute_indicators</td>
<td>Performs mathematical functions and inference for computing analysis indicators</td>
</tr>
<tr>
<td></td>
<td>Get_user_analysis_indicators</td>
<td>Retrieves the list of indicators to analyze the work of a particular user</td>
</tr>
<tr>
<td></td>
<td>Get_group_analysis_indicators</td>
<td>Retrieves the list of indicators to analyze the work of a group of users</td>
</tr>
<tr>
<td></td>
<td>Store_indicator</td>
<td>Records a set of indicators</td>
</tr>
<tr>
<td>Intervention</td>
<td>Visualize_advice</td>
<td>Displays advice messages (e.g. “improve your communication”, “reply to your colleagues faster”, etc.)</td>
</tr>
<tr>
<td></td>
<td>Visualize_indicators</td>
<td>Displays analysis indicators (e.g. the assessment of the coordination between users)</td>
</tr>
<tr>
<td></td>
<td>Change_tool_mode</td>
<td>Changes the mode for using a tool (e.g. to replace a proposal-based control policy with a democratic policy)</td>
</tr>
<tr>
<td></td>
<td>Change_user_interface</td>
<td>Changes an aspect of the user interface (e.g. modify the aspect of a workspace or change the tools it includes)</td>
</tr>
</tbody>
</table>

provides functions to be invoked in order to get information about the presence of users and their work. Finally, the Observation, Abstraction and Intervention subsystems provide the main operations for automating the three phases of the analysis life cycle.

4. Case study: COLLECE

The concern-based architecture proposed in the previous section was followed to integrate collaboration and interaction analysis mechanisms into the COLLECE [11] groupware system, which supports synchronous distributed collaborative programming practices, so that users located at different workstations can simultaneously collaborate in the creation of a computer program. The following subsection describes the COLLECE system from different perspectives. Section 4.2 shows in detail the process in which the proposed concern-based architecture is used to run a collaboration and interaction analysis within the COLLECE system.

4.1. COLLECE description

The framework for describing software-intensive system architectures defined in the IEEE 1471 standard [28] has been taken as a basis for describing the COLLECE system. This framework proposes the use of views to represent particular aspects of a system. However, there is a lack of standardized viewpoints (i.e. the conventions for creating, analyzing and depicting a view) to be used for modeling the system. Consequently, a wide range of different viewpoints has been proposed for the architectural modeling of software systems [36]. Our study makes use of those viewpoints that are common to all of the proposals, as they allow for the modeling of the main system components, their functions and the system execution. The considered viewpoints are:
4.1.1. Conceptual viewpoint

This view shows the basic functions that the COLLECE system aims to provide its users with. The main functions of COLLECE focus on supporting distributed synchronous collaborative programming. The users of such functions play the role of programmers: they collaborate with other users playing the same role, located at other workstations. COLLECE therefore provides shared workspaces that are used to solve programming problems that the users are set. Fig. 3 shows the conceptual model of COLLECE, modeled as a set of shared workspaces (edit, compilation and execution) and a set of communication flows established between them and the users. This conceptual model also shows how the COLLECE system integrates a collaboration and interaction analysis subsystem that supports collaboration and interaction analysis mechanisms (see Section 4.2 for a description of these mechanisms). The interactions generated by the programmers’ work in the shared workspaces, according to the specification of the analyst, are collected and stored in a repository.

4.1.2. Functional viewpoint

COLLECE provides a user interface that enables each programmer to interact with the groupware system through a set of tools in shared workspaces and build the required computer program collaboratively. These collaborative tools are:

- Code editor (Fig. 4-a): This is a shared editor in which the program source code is written.
- Structured chat (Fig. 4-b): This tool is used for communication. Although it can be used just like a traditional chat tool, it also contains a set of pre-defined communicative acts (e.g. “I think that …”) to make communication more fluid. In order to communicate ideas to other participants, the user clicks a button containing the label identifying the communicative act and then simply completes the sentence.
- Coordination panels: These tools enable the coordination of program editing (Fig. 4-e), compilation (Fig. 4-d) and execution (Fig. 4-f). The users use buttons on the corresponding panels to request participation in a process and to accept or reject such requests. If a user’s request for editing, compiling or executing is accepted by all the group members, that user takes control of the floor and can use either the editor (Fig. 4-a) to code, or the console (Fig. 4-c) to compile or execute. Apart from this proposal-based coordination model, these tools support the democratic and request/release models.

Moreover, the Application subsystem processes the information provided by the Awareness subsystem to show detailed information about the work performed by other programmers in the group in the following panels:

- Session panel (Fig. 4-g): This contains information about each user: (i) photo, (ii) a description of the user’s activity (i.e. editing, compiling, etc.) and (iii) the user’s name in the same color as the user’s tele-pointer.
- Tele-pointers (Fig. 4-h): These are marks (colored rectangles) in the editor that highlight the line of code being edited by a user.
- Group state label (Fig. 4-i): This text describes the group’s work phase (editing, compilation or execution).
- Semaphores: These “lights” are incorporated into the coordination panels (Fig. 4-e, 4-d, 4-f). The possible colors are: green, when there is a request waiting to be answered by the user; and red, when there are no requests pending in the corresponding panel.

Finally, the Application subsystem provides support for performing individual actions. These individual actions are only perceived by the user who performs them and they do not modify the state of the shared workspaces or the contextual panels. These actions are as follows:

- Reading of the problem (Fig. 4-j): This functionality opens up a window showing textual information about the requirements to be met by the computer program.
- Statistics (Fig. 4-k): This functionality opens up a window showing some statistics about the compilation processes.
• Analysis (Fig. 4-1): This functionality opens up a window showing quantitative and qualitative indicators analyzing the collaborative work carried out by the programmers.

• Individual editing actions: The Application subsystem provides the usual features of any regular text editor (e.g. cut, copy, paste, open, save, etc.). Some of these functions are only perceived by the user who performs them. This is the case with storing the program or copying an excerpt of source code to the clipboard.

4.1.3. Execution viewpoint

This view maps the COLLECE system onto the runtime platform and specifies the communications between its components. COLLECE was implemented using the Java language and operates through a client/server approach where the clients are the users’ applications, synchronized by a central server.

The user accesses the COLLECE system through the web browser. The client application then launches the user interface to provide the services required by the user according to his/her role in the collaborative system. All events performed within the user interface (e.g. changes introduced in a workspace by the programmer) are reported by the client application to the COLLECE synchronization server. This server is at a distinct physical location to the web server since both are independent. In the same way, the centralized database is deployed on a separate server in order to provide improved system performance.

The COLLECE client and the COLLECE synchronization server exchange information through sockets services. The Infrastructure for Collaborative Systems Synchronization (ICSS) has been developed to provide functionality for synchronization of shared workspaces in real time [5]. This infrastructure was built using the Java Shared Data Toolkit (JSDT) [47]. JSDT not only allows the groupware system to exchange data via a client/server approach, but also implements a lightweight reliable multicast protocol to support collaboration in real time.

4.2. Applying the concern-based architecture in order to integrate analysis mechanisms

The COLLECE system was developed following the architecture described in Section 3. This architecture proposes the development of a set of subsystems with well-defined functionality within the system. In the COLLECE system, the Identification subsystem is responsible for identifying the role of the user (programmer or analyst) and for launching the Application subsystem that supports the functionality to be used according to the user’s role. A user is defined mainly by three descriptors: name, password and the role played in the collaborative system. Each work session is defined by a set of programmers, the programming task to be carried out by them and the timetable. The Meta-information subsystem manages information about the configuration of the shared workspaces, including the floor control policies for the coordination panels in each session (i.e. the proposal-based, democratic or request/release coordination protocol) and the conversational acts to be included in the structured chat tool.

The Application subsystem manages the user interface that supports the functionality required by each user according to his/her role within the collaborative system. In the COLLECE system, when the user accessing the system plays the role of
programmer, he/she is placed into a working session by the Application subsystem. Each session provides the programmer with a set of partners to collaborate with, along with the task to perform or the problem to solve. The Awareness subsystem synchronizes the shared workspaces (source code editor, console, coordination panels and structured chat) and provides information for updating those components that show contextual information (session panels, state labels, tele-pointers, etc.). The users playing the role of analyst can access a list of the sessions provided by the Application subsystem and can then select a session to perform the analysis process on.

The Analysis subsystem integrated into COLLECE in accordance with the proposed architecture includes an Observation subsystem for processing models known as Observation models, which specify the actions to be captured and stored. These models are hierarchical structures with the following elements (Table 2) (see [13]):

- **Root element**: This identifies the groupware system whose use is the subject of the analysis, in this case COLLECE.
- **Collaborative tools**: These are the collaborative tools integrated into the Application subsystem; they are identified in the second level of the model.
- **Categories of actions**: These are third level nodes that characterize the actions supported by each collaborative tool according to analysis requirements. The different actions categories used by the Observation subsystem are as follows:
  - **Instrumental**: Actions that modify the state of the built product, in this case by modifications made to the source code, for example.
  - **Cognitive**: Actions that enable the validity of the built product to be tested, in this case compiling or running the implemented program.
  - **Communicative**: Actions that allow for communication (dialog) between users.
  - **Formal**: Actions enabling coordination and synchronization of users’ activity in the workspaces, commonly by means of floor control and decision-making tools.
- **Actions**: These are the names that refer to the actions to be selected by the Observation subsystem. Each name must match the name assigned to the action by the Meta-information subsystem.

Table 2
Nodes used in the observation models.

<table>
<thead>
<tr>
<th>Icon</th>
<th>Level</th>
<th>Node</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Icon" /></td>
<td>1</td>
<td>Groupware system</td>
</tr>
<tr>
<td><img src="image2" alt="Icon" /></td>
<td>2</td>
<td>Collaborative tool</td>
</tr>
<tr>
<td><img src="image3" alt="Icon" /></td>
<td>3</td>
<td>Instrumental action</td>
</tr>
<tr>
<td><img src="image4" alt="Icon" /></td>
<td>3</td>
<td>Cognitive action</td>
</tr>
<tr>
<td><img src="image5" alt="Icon" /></td>
<td>3</td>
<td>Communicative action</td>
</tr>
<tr>
<td><img src="image6" alt="Icon" /></td>
<td>3</td>
<td>Formal action</td>
</tr>
<tr>
<td><img src="image7" alt="Icon" /></td>
<td>4</td>
<td>Action identifier</td>
</tr>
</tbody>
</table>

Fig. 5 shows a fragment of the observation model designed by the analyst for filtering actions in COLLECE. The first level of the observation model includes a node specifying the groupware system that includes the Analysis subsystem. The second level comprises nodes that identify the collaborative tools built into COLLECE (structured chat, console, code editor and coordination panels) that support the actions to be filtered. This fragment of the observation model (see Fig. 5) shows the actions supported by the chat, the code editor and the execution coordination panel. The third level nodes describe the actions supported by each tool; for example, the chat tool supports communicative actions and some communicative acts are included ("I see a mistake", "I miss a ...", etc.). The editor supports instrumental actions such as inserting characters, deleting code, etc. The coordination panels support formal actions (agreements between programmers) and in the case of the execution coordination panel, cognitive actions (execution of programs). Finally, the fourth level assigns an identifier name to each action to be filtered.

The stored data that represent descriptors of the actions specified in the observation model are as follows:

- **Time**: When the action is performed.
- **User**: Programmer who performs the action.
- **Session**: Working session in which the action takes place.
4.2.1. Some experiences

An experience was carried out in order to put the architecture, and specifically its Analysis subsystem, into action and to evaluate the automated analysis with the help of external stakeholders (groupware users and analysts). Thirty-eight students participated in the experience as users of COLLECE. These students were from the School of Computer Engineering at the University of Castilla-La Mancha (Spain) and they each had at least one year’s programming experience. The students were randomly grouped into pairs and asked to solve a programming problem. The goal of the performed analysis was to evaluate the collaborative programming process by means of indicators aimed at analyzing the main aspects of the collaboration (amount of work, working speed, communication, coordination, etc.) and to intervene in the collaborative work when an analysis indicator reflected any kind of problem. The students were informed that their activity would be observed for collaboration and interaction analysis purposes. Moreover, they were told that the privacy policies followed do not allow the data captured about their activities to be passed on to third parties.

Seven computer engineers playing the role of analysts, defined a collaboration and interaction analysis process to observe and to characterize the collaborative work of the students and consequently to intervene in the group activity. The interventions defined by the analysts were classified into two categories: visualizations and adaptations. A visualization would display a message advising the students how to continue with their work showing analysis indicators characterizing particular aspects of the collaboration. An adaptation consisted in replacing the predefined messages included in the structured chat by others that better fit the current status of collaboration and in choosing the most suitable floor control policies for the coordination panels when they hindered coordination between users. Fig. 6 shows an example of a visualization, which both displays an analysis indicator informing the user about the low number of completed executions and advises them to test the program more often by executing it.

Fig. 7 shows a UML sequence diagram that models the interactions of the COLLECE subsystems when cooperating in an adaptation intervention. It shows how an analyst interacts with the Analysis subsystem to associate an observation model with a COLLECE session. The students playing the role of COLLECE programmers in the session then access and interact in the shared workspaces that are provided by the Application subsystem. This interaction is illustrated in Fig. 7 with a request by the programmer to execute the program built. This request generates calls to (i) the Meta-information subsystem to test the user permissions to perform the action, (ii) the Awareness subsystem to communicate the request to other programmers, and (iii) the Analysis subsystem to filter and store the interaction information according to the observation model. Lastly, the analyst interacts with the Analysis subsystem to visualize information about the collaborative process and to change the chat functionality to free mode. In those cases in which this visualization shows a low level of communication, the analyst
would interact with the Analysis subsystem, which would invoke a method of the Meta-information subsystem that allows the messages available in the structured chat to be changed, in order to enable free communication without pre-defined messages.

After finishing the collaborative programming experience, the students answered a questionnaire, giving us their opinions on the analysis process. The first question asked them to evaluate the advice-type interventions. The students were asked about the suitability of the advice provided by the Analysis subsystem. The second question went deeper into the assessment of the suitability of the interventions that involved visualizations of indicators characterizing the collaborative work. The third question tried to find out whether the interventions related to the adaptation of the groupware (replacing predefined messages in the structured chat and changing floor control policies) are a suitable way to improve their work. Finally, the fourth question asked the students for their opinions on the global intervention as the result of the analysis process. The students answered each question with a five-point Likert scale ranging from 1 (worst score) to 5 (best score). Table 3 contains a statistical summary of the scores from the questionnaire as answered by the students. This table shows that most students gave the various aspects of the questionnaire a good evaluation (4 in a five-point scale).

The second group of external stakeholders was made up of the seven analysts responsible for defining the analysis process. They took part in a survey designed to obtain an assessment of the support provided for defining the analysis process. The survey consisted of three questions. The first one asked the analysts to give a general score to the model's suitability for specifying the actions to be captured in the observation phase (observation model). The second question asked for an evaluation of the suitability of the interventions that show analysis indicators and advice, and which adapts the structured chat and coordination panels. Finally, the third question was a global evaluation of the entire analysis process. As before, the analysts answered each question using a five-point Likert scale. Table 4 shows a statistical summary of scores from the evaluation completed by the analysts. In each case, the intermediate (3 in a five-point scale) and high values (4 in a five-point scale) were commonly chosen by the analysts when evaluating the different aspects of the questionnaire.

### Table 3
Students' opinions.

<table>
<thead>
<tr>
<th>Question</th>
<th>Mode</th>
<th>Median</th>
<th>Range</th>
<th>Inter-quartile range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assessment of the advice</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>0.5</td>
</tr>
<tr>
<td>Quality of the indicator</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>0.5</td>
</tr>
<tr>
<td>Suitability of the adaptations</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Global evaluation of the analysis</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

### Table 4
Evaluation by analysts.

<table>
<thead>
<tr>
<th>Question</th>
<th>Mode</th>
<th>Median</th>
<th>Range</th>
<th>Inter-quartile range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assessment of the observation mode</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>0.5</td>
</tr>
<tr>
<td>Quality of the interventions</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>0.5</td>
</tr>
<tr>
<td>Global evaluation of the analysis process</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
5. Evaluation of the architecture

After having defined an architectural model that allows groupware systems that incorporate a collaboration and interaction analysis subsystem to be easily designed, and having applied the proposed architecture to the COLLECE groupware system, the next challenge is to evaluate the architecture as a means for facilitating the integration of analysis mechanisms into groupware systems of different characteristics. Kazman et al. [31] proposed the SAAM (Software Architecture Analysis Method) method to assess the attributes of software architectures. This method aims to evaluate the attributes of an architecture through its use in different scenarios. The method classifies scenarios as either direct or indirect. In the context of collaboration and interaction analysis, the direct scenarios are those where the architecture fits in perfectly with the requirements of the analysis process that is to be automated. The indirect scenarios are those where the architecture must undergo either major or minor changes in order to meet the requirements. This way, SAAM serves as a basis to measure the flexibility (using the definition extracted from [27]) of the proposed architecture.

Our architecture has been used in three different scenarios each of which differs from the scenario that COLLECE represents (see Section 4) in certain basic characteristics. The first scenario consists in the use of a groupware system that supports the collaborative designing of models. The second scenario deals with a groupware system that supports asynchronous collaboration. Finally, the third scenario involves the integration of analysis support into mobile groupware systems. Specifically, the goal is to study the suitability of the architecture in these scenarios with regards the phases of observation and intervention, by means of visualizations and adaptations.

In the first scenario, the architecture was used to integrate an Analysis subsystem into SPACE-DESIGN [12], which supports synchronous distributed collaborative modeling practices. The analyst designed specific analysis indicators for evaluating collaborative modeling work and defined interventions according to the characteristics of SPACE-DESIGN. We verified that the proposed architectural structure supported the automation of the analysis process [13] and it was not necessary to include new components or modify the functional relations between them.

The construction of both the SPACE-DESIGN and COLLECE systems with integrated analysis functionalities allowed for the evaluation of the architecture’s flexibility in terms of its being adapted to different systems that support synchronous distributed collaboration. The structure of the architecture takes advantage of the independence between the Analysis subsystem and the other components that focus on managing the users’ activity. Therefore, the indicators that must be inferred and the interventions that must be performed are specified by means of the Analysis subsystem without affecting the components that manage the users (Meta-information, Awareness and Identification subsystems).

In the second scenario, analysis support was integrated into PlanEdit [42], a groupware system that supports the asynchronous distributed collaborative building of action plans in the field of Domotics (i.e. house automation), using the architecture as a base. Asynchronous means that not all the users are working simultaneously (see the time-related interaction type concern in Section 3). For this reason, analysis indicators and intervention mechanisms that were specially oriented to the asynchronous planning task of PlanEdit were selected and designed [14]. Our conclusion, therefore, is that the architecture makes a suitable base for designing an asynchronous collaboration analysis subsystem, making use of the variabilities of the architecture (as in the previous scenario), and that there is no substantial difference with synchronous collaboration.

The construction of the SPACE-DESIGN, PlanEdit and COLLECE systems with integrated analysis subsystems allowed for the evaluation of the reusability of the architecture in cases of synchronous and asynchronous collaboration. Despite the different nature of the collaboration, the architecture enabled the integration of the Analysis subsystem because it includes a Meta-information subsystem that captures information about the users’ activity regardless of the type of collaboration.

Finally, the third scenario considers the challenge of incorporating analysis support into mobile groupware systems. First of all, the proposed architecture focuses mainly on integrating analysis subsystems into traditional groupware systems that support different kinds of collaboration and application domains. Some limitations should be considered in the case of collaboration between mobile users. The architectures and functions of mobile groupware systems usually involve specific constraints [39]. For example, in many cases information is synchronized on-demand; in other words, there is no up-to-date central repository containing all of the user interactions at any given time because this repository is only updated when the user is connected. Therefore, the analysis processes would have to work with incomplete information since not all of the interactions, representing the users’ activity, would be available. Similarly, the intervention could have a limited effectiveness when there are users working off-line. Also, the synchronization process, which is carried out when the users obtain a connection to the system, should include the synchronization of the past users’ interactions and the updating of the group analysis indicators.

At this point, it should be noted that the developer must implement variabilities that affect not only the Analysis subsystem but also other subsystems of the architecture. On the one hand, the developer must consider the development of variabilities in the Meta-information subsystem to identify the users who had worked off-line and whose interactions could not be communicated to the Analysis subsystem in real time. In addition, the developer must implement new analysis variabilities to design indicators and interventions oriented towards off-line work.

Summing up, the proposed architecture has been evaluated in terms of its application to three different scenarios. The first scenario, a synchronous collaborative modeling system, is considered a direct scenario since the architecture need not be changed. The second scenario, an asynchronous system, is also considered to be a direct scenario because no changes are required in the architecture, even though it is necessary to define specific analysis indicators and ways of intervening
to deal with the fact that the users will not necessarily be working at the same time. Finally, the mobile groupware system scenario is an indirect scenario since major changes are required to the architecture in order to handle situations in which users may be working off-line and their interactions will not be processed in real time by the Meta-information subsystem.

6. Related work

In recent years, the challenge posed by increasing business process productivity has led to the analysis of collaborative work by means of process mining techniques that extract knowledge from log documents generated by groupware systems [24]. This is the case of the use of algorithms to analyze so-called workflow logs [51]. In this same line, van der Aalst et al. [50] proposed a standard workflow log specification, but this proposal lacks an architectural design that can guide developers when integrating observation mechanisms into information system. These authors assume that the functionality of detecting and storing events in groupware system is implicit and they do not provide any architectural model that would enable the collection and selection of actions to be stored in workflow logs.

A special case is that of Process-Aware Information Systems (PAISs) which integrate mechanisms to support the collaboration of different systems and users. According to Dumas et al. [10], a PAIS can be defined as “software as a system that manages and executes operational processes involving people, applications and/or information sources on the basis of process models”. PAISs should therefore allow for coordinated processes through the collaboration of users of different software applications. As such, a PAIS should support observation mechanisms that detect user actions and assess their impact on the work process according to a process model. This challenge is partially addressed by the Caramba PAIS [15], which implements a middleware architecture supporting virtual teams made up of enterprise applications. When a user generates meaningful information through various tools (e.g. web browser, Java client application, etc.) for other users, the application sends this information to the Caramba system for distribution. However, this proposal does not automate the analysis process that intervenes in the collaborative improvement work.

In the field of CSCL, several proposals have designed systems for analyzing user interactions. The Elluminate commercial system [17] enables the automatic production of collaborative virtual learning environments. Elluminate automatically collects learners’ interactions, however it does not include mechanisms for inferring analysis indicators or for intervening in the collaborative activity. Martinez et al. [35] classify the architectures used by the developers of CSCL environments which include analysis support into three categories: (i) black-box architectures, where the developers cannot access the source code of the collaborative system, (ii) white-box architecture, where the developers can access the source code and build ad hoc solutions, and (iii) interaction-aware decoupled architectures, which guarantee the collaborative system’s independence from the analysis system, with intercommunication taking place through messages or log documents. Harrer et al. [25] conducted more in-depth studies of interaction-aware decoupled architectures, proposing two basic components: (i) adapters, which are interfaces between the collaborative system and the integrated component (i.e. the analysis subsystem), and (ii) Composite Data Structures, which enable access to certain parts of the data shared between the collaborative system and the analysis subsystem. A Composite Data Structure supports subscription mechanisms for sending the analysis subsystem only certain data when this structure changes in the collaborative system. Mühlenbrock et al. [38] proposed a technological infrastructure to design a distributed architecture supporting a collaborative learning environment, the Dalis architecture [37], which contains an analyzer for evaluating user interactions in the shared workspaces. Since the Dalis architecture does not incorporate the analyzer into the groupware system, additional modules must be installed to capture any interactions that are performed and to communicate them to the analyzer.

To sum up, there does exist an initial set of proposals in the fields of business processes and workflows which are capable of characterizing the users’ activity using process modeling techniques. However, given the challenge described in Section 4 (i.e. the integration of analysis mechanisms into a groupware system), these approaches do not put forward any architectural model to make this integration possible. In contrast, a second set of proposals in the fields of CSCL systems and PAISs do introduce architectural considerations. The PAISs allow information about each user’s interaction to be collected and stored. Some CSCL environments propose architectural designs to provide separation between the groupware system and the system that performs the analysis process. However, these proposals do not offer a reference model that identifies subsystems or main components through which to automate the analysis process by means of observation of the users’ work and the intervention in this work.

7. Conclusions

This paper presents as its main contribution a concern-based architecture which enables developers to design groupware systems which integrate a collaboration and interaction analysis subsystem. This architecture specifies the entire groupware system through a set of subsystems (Identification, Meta-information, Awareness, Analysis and Application) which interact with each other to support the basic functions of any groupware system and to analyze the collaborative work carried out by the users. These basic functions include user management and providing support for collaborative tasks within shared workspaces. Furthermore, the architecture allows for the analysis of actions not only for inferring analysis indicators and displaying advice but also for dynamically improving the collaborative work process by adapting the system according to the different pre-configured system behaviors that are available. As such, the Analysis subsystem is made up of an Observation
subsystem that captures and records the users’ actions, an Abstraction subsystem that infers analysis indicators and an Intervention subsystem aimed at improving the collaborative work through different kinds of actions. A set of commonalities or common features can be derived by all of the software systems built using this architecture. The source of these commonalities is a common architecture that includes subsystems for identifying the groupware users, for managing information about the collaborative process, for creating awareness information to allow for the updating of the shared workspaces of the Application subsystem, and for analyzing the collaborative work in an observation–abstraction–intervention life cycle. These commonalities are complemented by some variabilities or specific features that should be implemented to infer suitable analysis indicators for a specific collaborative process and to intervene according to the particular characteristics of the groupware system in question.

The architectural model presented here was used to develop COLLECE, a collaborative programming system, and to integrate into it a subsystem for collaboration and interaction analysis. COLLECE was used in an initial set of experiences to automate the analysis of collaborative programming practices and to implement the reconfiguration of the system (communication facilities of the chat and decision-making model of the coordination tools). In these experiences, some stakeholders, i.e. the users and analysts of COLLECE, gave a positive assessment of the suitability and the quality of the three automated phases (observation, collaboration and intervention) of the analysis process. In addition, using the architecture to integrate analysis mechanisms into other groupware systems, of different characteristics, has allowed us a discussion of the features provided by the architectural model. Specifically, the proposed architecture has fit the analysis requirements of both synchronous and asynchronous distributed collaborative scenarios and those of collaborative tasks of different types (e.g. structured problem solving or open discussions).

The architecture has also presented some limitations. Firstly, the architecture needs improvements to be made in order for it to meet the requirements of certain scenarios such as collaboration supported by mobile groupware systems. In these cases, the Meta-information subsystem does not have access to an up-to-date repository containing data about the users’ activities. The flexibility of the architecture is limited in these cases because it does not fit in with the requirements of scenarios in which the Analysis subsystem is unable to capture the users’ actions in real time. Secondly, the analysis is limited in certain scenarios such as those in which the participants collaborate using the same workstation, since the face-to-face communication actions cannot be captured by the Analysis subsystem, or scenarios of asynchronous collaboration, where not all of the users work simultaneously and the Analysis subsystem therefore cannot improve aspects of the collaboration for which the simultaneous participation of all of the group members is required.

There are many groupware systems available on the market. As a non-exhaustive list of examples we can mention real-time strategy games, collaborative board games (e.g. chess games), CSCL environments for different domains, collaborative modeling systems (e.g. CAD/CAM systems), electronic meeting systems, collaborative software development environments, and so on. We think that the proposed architecture and the underlying method could be applied to all of these systems in as much as they enable the design of an analysis subsystem, for different purposes, which could be integrated into each of them.

This novel approach is being validated by means of experiments which set out to evaluate the differences between the integration of analysis support into groupware systems using our proposal and the integration of analysis support following traditional approaches, in which the analysis mechanisms are designed from scratch and without any reference model. Our main goal is to compare aspects such as the time spent on the implementation and integration processes, the quality of the developed analysis subsystem, and the opinions of the developers and groupware users. Our future efforts will be aimed at completing the proposed architecture with mechanisms for automating the abstraction and intervention phases of the analysis life cycle using computational models. The current approach uses computational models only to define the capturing and filtering of actions during the observation phase. Our objective will be to reduce the effort needed to define and compute analysis indicators and to specify and execute interventions in the collaborative work.

Acknowledgements

This research has been partially supported by R + D projects of the Spanish MCYT under projects TIN2007-60199 and TIN2008-05995/TSI. This research has been partially supported by the Junta de Comunidades de Castilla – La Mancha (Spain), Consejería de Educación y Ciencia, under Projects PAC07-0020-5702 and PCI08-0069-7887.

References


