A Distributed Intelligent Pair-Software Development Tool

Eustáquio São José de Faria[1], Keiji Yamanaka[1], Josimeire do Amaral Tavares[2], Geraldo Henrique Lacerda Pinto[2] and Lowghan Henrique Sudário de Melo[2]

[1] Faculdade de Engenharia Elétrica
Universidade Federal de Uberlândia – Uberlândia, MG – Brasil

[2] Instituto de Informática
Pontifícia Universidade Católica de Minas Gerais – Arcos, MG – Brasil

eustaquio@pucminas.br, keiji@ufu.br, prajosy@yahoo.com.br, goblinbr@gmail.com, lowghan@yahoo.com.br

Abstract. *Several researches about pair programming have been developed defending its viability and efficiency on quality software development practice. In many of these studies, the Pair Programming was accomplished in a co-located way and, in others, in a distributed way. In this paper, it's intended to develop a distributed intelligent pair-software development environment to support co-located or distributed pair programming. Activities like objective declaration, use cases, use-case descriptions, activity diagrams, test cases, class diagrams and program coding can be done by the pairs using this software-development environment. Voice communication is also allowed. Intelligent agents were developed intending to mediate the team members' collaboration. The system is in its test phase in an empirical study in a Information System Course.*

1. Introduction

Pair Programming is a software production practice adopted by a rapid development methodology known as Extreme Programming. Such practice has become quite viable and efficient on medium and huge-quality software development projects.

During the software development process, the programmers' pairing can be done in a distributed way (using groupware tools) or in a co-located way (in witch two programmers get together in front of the same workstation, side by side).

In a previous study, Faria and Yamanaka (2008) described some groupware tools available which could be used as aid to distributed Pair Programming. In addition, several studies have been researched (BAHETI *et al.*, 2002, BAHETI, GEHRINGER and STOTTS, 2002, BRYANT, ROMERO and BOULAY, 2005, HANKS, 2002, HANKS, 2004, STOTTS *et al.*, 2003, ZIN, IDRIS and SUBRAMANIAM, 2006) in witch the authors evaluated the effectiveness of commercial tools with such purpose, such as: Skype, Microsoft NetMeeting, PC Anywhere, VNC, Yahoo Messenger, PalTalk, AOL Messenger, Learning Manager System, among others.

In addition, specific tools to support Pair Programming were also developed. Langton, Hickey and Alterman (2004 and 2005) developed and tested a tool formerly named GHT, and nowadays known as GREWPtool; Atsuta and Matsuura (2004) created a pair-programming distributed environment and proposed necessary improvements so that project managers could also use such tools in their managing functions; A system named SANGAM was described in Ho *et al.* (2004) – this system supports pair programming, but doesn't support paired-project process; A computer-aided collaboration environment named GILD was developed in Cubranic and Storey (2005) and Cubranic, Storey and Ryall (2006) – the system allows programmers' pairing, although it doesn't offer support on communication, which is done using commercial tools like Instant Messengers, Skype, among others. The study describes the necessary requirements to develop collaborative environments; A unique methodology was proved in Baheti *et al.* (2002) – based on this study, a collaborative game was developed to teach Java concepts. In this game, the players think, in groups, about specific problem solutions and then are paired to solve them. Later on, other exercises give the students opportunities to work with their own problems; DeFranco-Tommarello and Deek (2005) developed a cognitive collaboration model named CCM – in this paper, the authors provide a tutorial teaching how to solve collaborative programming problems and describe a collaborative tool developed under the model's concepts; Mendes *et al.* (2005) developed the COLLEGE, a collaborative programming environment which supports Pair Programming – the authors tested the system with students from Coimbra University, but didn't describe the experiment results; Natsu *et al.* (2003) developed a synchronous shared source-code editor named COPPER (COllaborative Pair Programming EditoR) which allows the distributed software paired coding. The system implements groupware-environment characteristics such as communication mechanisms, collaboration consciousness, concurrent control, among other features.

Most of the researches aforementioned applied distributed pair programming intending to compare it to the co-located pair programming. Results from such researches are highlighted below:

- Co-located pairs haven't reached, significantly, better results than distributed pairs (BAHETI *et al.*, 2002 and BAHETI, GEHRINGER and STOTTS, 2002);
- Distributed pair programming results in a code which is of same quality compared to a co-located one (STOTTS *et al.*, 2003);
- Distributed pairs keep many of the benefits (pair pressure, knowledge sharing, among others) related to co-located pairs (STOTTS *et al.*, 2003);
- Although it isn't statistically significant or conclusive, distributed pairs showed better results on final exams compared to co-located pairs (HANKS, 2004);
- Distributed pair-programming tools provide important information related to communication, work history, actual status, elapsed time, etc., either to project managers or to teachers (ATSUTA AND MATSUURA, 2004);
- Communication is the main factor for the pairing-process success. The process of software development fails when the distributed pair-programming tool doesn't provide communication and collaboration effectively (CANFORA *et al.*, 2006 and VISAGGIO, 2005).

In an empirical study with distributed pair programming, Hanks (2006) raised an important question: Can teachers, as mediators, influence both the students' attitudes towards the pair programming process and the general learning in paired environments? - It’s believed that it's a strong influence, since the human mediation is restricted to availability, it's defended that intelligent agents can be efficient to mediate the collaboration between the developers, fulfilling the teachers' and project managers' activities.

Canfora *et al.* (2006) approved the use of distributed pair-programming environments and described the necessary characteristics so that they can be truly effective.

According to the authors, systems with such features must allow audio communication, videoconference and code sharing.

Important hints for the good working of distributed pair programming can be found in Stotts *et al.* (2003). The authors discussed basic guidelines and concluded that the greatest problem in distributed pair-programming environments is related to communication between its participants. It was suggested that these environments be provided with audio resources which allow communication, for instance, with Voice over IP.

Some of the pair-programming available tools in the literature don't support internal compilation of the coded programs and, apparently, none of them allows voice communication. It's also important to emphasize that no pair-programming applications with Artificial Intelligence features were related or found. Thus, no paper on Pair Programming has described the existence of systems which support paired projects, tests and reviews (although the research wasn't focused on such purpose). It's believed that those are strong reasons to justify the development of this study, whose main objective is to show the project and development of an intelligent distributed collaborative pair-programming computer software development environment.

The following section describes the declaration of the proposed system's objects. Section 3 shows the use-case diagrams and its descriptions and section 4 provides the conclusions and suggestions for future projects.

2. Aiddes Declaration Of Objectives

Based on several problems faced by the traditional education in the teaching of algorithms and programming techniques and other reasons previously mentioned, it's intended, in this article, to produce an intelligent and collaborative pair-programming computer software development environment denominated AIDDES (in Portuguese: Ambiente Inteligente Distribuído de Desenvolvimento Emparelhado de Software). The environment is formed by a synchronous intelligent coding editor and a CASE tool, both shared, in which users can develop the following activities: (1) declaration of objectives; (2) use cases; (3) description of the use cases; (4) activity diagrams; (5) test cases; (6) class diagrams; and (7) source-code implementation. AIDDES also allows voice communication, by either Chat or discussion Forum, during the accomplishment of the pairing sessions.

Two fields can be distinguished on AIDDES: (1) Administrator’s area; and (2) Developer’s area.

The Administrator accompanies the discussions through Chat and Forum, participating when necessary, and s/he is also responsible for inviting non-registered users (it is important to point out that only users invited by the Administrator can indeed use the system). Administrators are teachers or project managers.

The developers are the programmers, students or professionals, formerly invited by the Administrator. They should invite other users (already registered in AIDDES) to form development teams, also known as pairing sessions – when beginning a pairing session, the Driver should acknowledge whether the program will be developed in one of the languages that the environment supports (C, C++, Java, Fortran 77, ADA 95 or Pascal) or other language. If the driver chooses anyone of above mentioned languages, he should register the key-terms used by the ontological evaluators. The teams form a pairing session which is not related to other teams; therefore, a developer can compose several teams at the same time. Unregistered developers must fill out an online form in AIDDES requesting, to the Administrator, their registration in the system. The developer's roles are the commonly known ones in the literature: the Driver and the Navigator. Initially, the system gives the developer responsible for the partnership invitation the Driver's role, whereas the invited user is considered the Navigator. The roles can be constantly exchanged with agreement of the team's members. All the activities (Project, coding and tests) are paired-developed. Any changes (edition) on the activities in a pairing session is only allowed if both of the team members are connected to the system at that time – this helps to avoid any individual activity. At any time during the process the developers are able to inform to the system that the task is finished. From this moment on, changes are only allowed if both members request, again, for permission to reopen the task (pairing session reactivation) for production. It is important to point out that a pairing session is related to an only task to be accomplished, which is composed by its seven activities.

Intelligent Participation Agents are responsible for verifying, based on ontologies, the quality of the discussions stored in Chat and Forum data during a pairing session. It is important to point out that all the discussions through Chat are also stored in the system and can be seen by the pairs anytime during the development. Interventions are made by the agents, when necessary, intending to improve both the discussions and the participation relevance rate of each member. In addition, the agents are responsible for suggesting the constant change of roles. There are four types of software agents available in AIDDES: "Intelligent Agents for Verifying the Relevance of Comments and Identifiers", "Intelligent Agents of Mediation", "Ontological Program Evaluator Agent", and "Ontological Discussions Evaluator Agent".

The "Intelligent Agents for Verifying the Relevance of Comments and Identifiers" are helped by an Ontological Program Evaluator (AOP). The AOP is a software agent that evaluates the student's program to verify the adaptation, according to ontologies, of the comments in the source-code and the mnemonics used for declaring variables, constants, data structures and functions' and procedures' names. This program intends to measure the ability of the teams' members in creating mnemonic and comments that fit accordingly in the context of the proposed task. To evaluate it ontologically, it is necessary to register the key-terms related to the task to be solved – this is done by the Driver during the creation of a pairing session or at any moment before the task is considered to be concluded.

The "Intelligent Agents of Mediation" actors are helped by an Ontological Discussion Evaluator (AOD), which intends to mediate the process of collaboration. AOD is a software agent similar to AOP, although AOD works evaluating the stored Chat and Forum discussions. This evaluator tries to measure the ability of the teams' members in participating of the discussions related to the context of the proposed task.

Information about the team members' elapsed time in each role (Driver and Navigator) are logged during the whole pairing session.

3. Aiddes Use-Case Diagram

For a better comprehension of the model, the use-case diagrams were subdivided according to the two great areas of the software environment: (1) Administrator's Area - Figure 1; and (2) Developer's Area (Drivers and Navigators) - Figures 2 and 3. In addition, the use cases diagrams that express the intelligent agents' actions were designed separately and they can be seen in Figure 4.

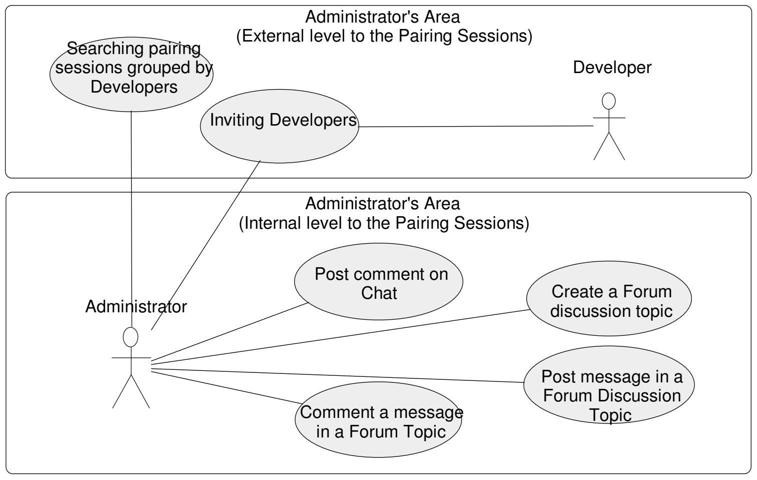
Administrator's Area
(External level to the Pairing Sessions)
Searching pairing sessions grouped by Developers
Inviting Developers
Developer
Administrator's Area
(Internal level to the Pairing Sessions)
Post comment on Chat
Create a Forum discussion topic
Administrator
Post message in a Forum Discussion Topic
Comment a message in a Forum Topic

Figure 1. AIDDES Use-Cases – Administrator's Area.

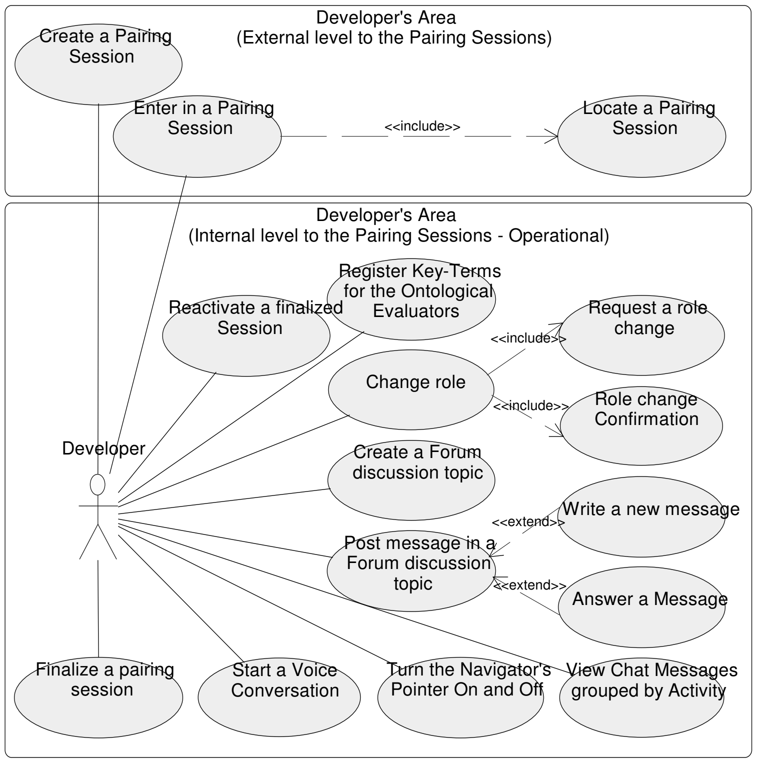
Developer's Area
(External level to the Pairing Sessions)
Create a Pairing Session
Enter in a Pairing Session
<<include>>
Locate a Pairing Session
Developer's Area
(Internal level to the Pairing Sessions - Operational)
Register Key-Terms for the Ontological Evaluators
Reactivate a finalized Session
Request a role change
<<include>>
Change role
<<include>>
Role change Confirmation
Developer
Create a Forum discussion topic
Write a new message
<<extend>>
Post message in a Forum discussion topic
<<extend>>
Answer a Message
Finalize a pairing session
Start a Voice Conversation
Turn the Navigator's Pointer On and Off
View Chat Messages grouped by Activity

Figure 2. AIDDES Use-cases – Developer's Area (Operational Scenario).

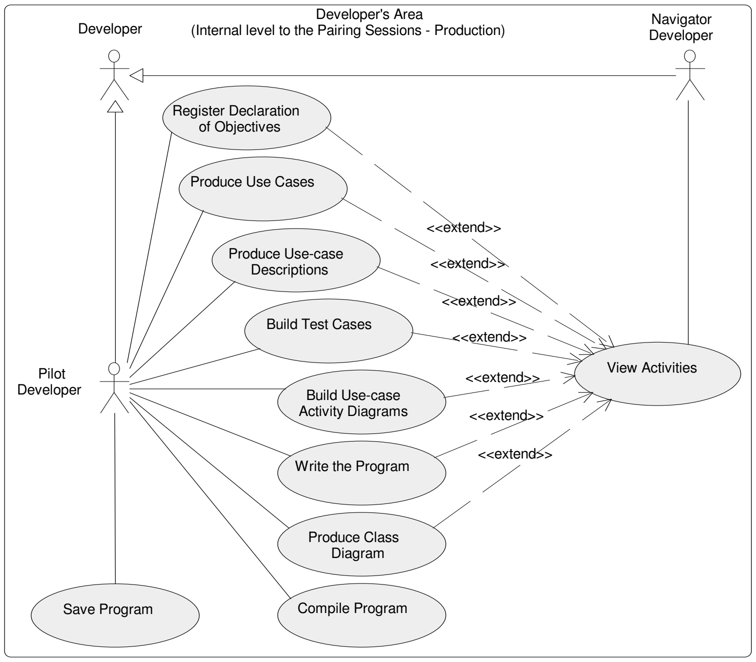

Figure 3. AIDDES Use-cases – Developer's Area (Production Scenario).

Constant automatic interventions are done by intelligent agents on the interactions between Navigators and Drivers, whenever it's necessary. According to Figure 4, agents can: (1) Intervene towards identifier contextual relevance – according to the Ontological Program Evaluator; (2) Intervene towards comments' contextual relevance – according to the Ontological Program Evaluator; (3) Intervene towards Chat messages' contextual relevance – according to the Ontological Discussion Evaluator; (4) Intervene towards Forum messages' contextual relevance – according to the Ontological Discussion Evaluator; (5) Intervene towards the amount of Chat contributions; and (6) Intervene towards the amount of Forum contributions.

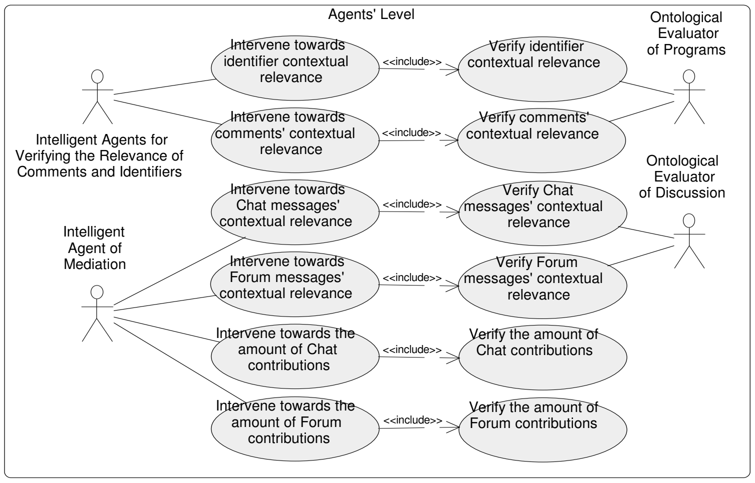
Agents' Level
Intelligent Agents for Verifying the Relevance of Comments and Identifiers
Intelligent Agent of Mediation
Ontological Evaluator of Programs
Ontological Evaluator of Discussion
Intervene towards identifier contextual relevance
Intervene towards comments' contextual relevance
Intervene towards Chat messages' contextual relevance
Intervene towards Forum messages' contextual relevance
Intervene towards the amount of Chat contributions
Intervene towards the amount of Forum contributions
<<include>>
Verify identifier contextual relevance
Verify comments' contextual relevance
Verify Chat messages' contextual relevance
Verify Forum messages' contextual relevance
Verify the amount of Chat contributions
Verify the amount of Forum contributions

Figure 4. AIDDES Use-cases – Agent Intervention Scenarios.

The table 1 describe AIDDES Use-cases except for the Agent Interventions' Scenarios, whose further details deserved its own article and can be found in Faria *et al.* (2008).

Table 1. AIDDES Use-Cases.

Area	Use-Cases
Administrator's Area	**Inviting Developers (Team members) -** Administrators send participation invitations and temporary passwords (via e-mail, using the environment) to the Developers requesting them to fill out the registration form in AIDDES. When the user signs in for the first time, AIDDES suggests that s/he change his/her password.
	Searching pairing sessions grouped by Developer - The Administrator, having the members' e-mail addresses and filling the edit fields, can locate a specific pairing session. This search can be done individually or grouped by pairs. A grid table with the search results and links allowing access to each pairing session's internal area is returned and shown on the screen.
	Post comment on Chat - After connected to a pairing session, the Administrator can comment Chat messages posted by the developers. These comments are important because they allow the Administrators to intervene in the coordination of the activities, as well as in the communication and collaboration among the team members.
	Create a Forum discussion topic - In a pairing session, whenever necessary, the Administrator can create discussion topics in the Forum intending to stimulate the social-cognitive conflict between the members of a team. Any forum topics, comments of Chat messages, message comments or Forum topics and messages posted by the Administrator are highlighted in red, whereas the others, created and posted by the Developers, are highlighted in blue.
	Post message in a Forum Discussion Topic - When connected to a pairing session, the Administrator can post messages in forum discussion topics. It's recommended that the messages posted by the Administrator have a questioning format. Answers posted by the Developers to such questions can give clues on how well is the communication, collaboration and coordination of the pairing-session activities.
	Comment a message in a Forum Topic - Besides the permission to comment Chat messages previously described, the Administrator can intervene in the pairing process commenting messages or Forum topics which s/he considers relevant.
Developer's Area	**Create a Pairing Session -** Any developer, after connecting to AIDDES, can invite another developer that's already registered at the system to compose a work team, forming a pairing session. After s/he accepts the invitation, the session is started and the inviter's status is activated as Driver, whereas the guest's is activated as Navigator.
	Enter a Pairing Session - After connecting to the system, the developers must require access to a pairing session. A list of sessions of which a developer is a member is shown in a grid table with access links to each one of them. After entering a session, the developer gets his role, the same one he had at the moment of his/her last access disconnection. In sessions considered finalized by both members, only a read-only access is allowed unless both members authorize, via AIDDES, a session reopening for updates.
	Locate a Pairing Session - Developers must, having his/her identification e-mail on AIDDES and using the search fields, locate a specific pairing session which s/he wishes to enter. This search can be done either individually or in pairs. A grid table is shown on the screen with the search results, containing access links to the internal area of each retrieved pairing session.
	Register Key-Terms for the Ontological Evaluators - The developers are allowed to register key-terms any time during an active pairing session. No activity (declaration of objectives, use cases, use-case descriptions, activity diagrams, test cases, class diagrams and coding) must be started without registering at least one key-term for that session.
	Change role - This event is triggered automatically after occurring, mandatorily, an event requesting a role change and an event confirming the role change confirmation.
	Request a role change - This event triggers a role change. It's important to point out that both the Driver and the Navigator are authorizes to request role changes anytime.
	Role change Confirmation - This event confirms or authorizes a role change and can only be triggered by the developer whose change was requested to.
	Create a Forum discussion topic - Discussion topics can be included any time as long as the session hasn't been finalized by both team members. It's important to point out that the Forum updates and the session finalization are the only active functionalities when only one of the developers is connected to the system.
	Post message in a Forum discussion topic - The developer must click a Forum message or topic and trigger the "Insert Message" button when either wishing to write a new message or answering a previously-posted message.
	Write a new message - New messages are those added in Forum topic. The inserting of a new message occurs when the user clicks a Forum topic and triggers the "Insert Message" button.
	Answer a Message - Message answers occur when the user clicks messages already posted in the Forum and triggers the "Insert Message" button.

(Continuing) Developer's Area	**View Chat Messages grouped by Activity -** The developers can, at any time and intending to increase the comprehension on the discussions, view all Chat messages classified according to the system's available activities (declaration of objectives, use cases, use-case descriptions, activity diagrams, test cases, class diagrams and coding).
	Turn the Navigator's Pointer On and Off - When there's need to focus on an important aspect in any of the activities, the Navigator can enable a pointer which is visible by the Driver in red. The Driver's pointer is always visible in blue by the team members. It's suggested that the Navigator disable his pointer at the end of his task.
	Start a Voice Conversation - The developers can communicate with each other verbally when triggering the "Voice" button. This feature, along with the pointing feature, allows communication almost equivalent to an in-loco one.
	Finalize a pairing session - The developers must trigger the "Finalize session" button where intending to consider the task finalized. Its status is only considered inactive after both the developers request the action. In addition, to change the task status to inactive, AIDDES demands that all the previous activities be finalized by both members.
	Reactivate a finalized Session - The developers must trigger the "Reactivate Session" button, when there's a consensus on the need of updating a task. The status is considered active only after its reactivation is requested by both developers. After reactivating the session, the status of all the activities keeps inactive until there's a change in one of the activities. Each modification changes the activity status from "finalized" to "in production".
	Register Declaration of Objectives - The activity of registering declarations of objectives is done with a synchronous shared editor that demands revision and acceptance of both members of the team so that it's considered finalized – the developers must trigger the "Finalize Declaration of Objectives" button. If any new character is inserted on the declaration of objectives, by the Driver, its status changes to "in production" again. This feature allows the developers to edit the declaration of objectives update, removing or inserting a new text fragment, any time. The developer in the role of Driver can update the declaration of objectives, whereas the Navigator can only view its information.
	Produce Use Cases - This feature makes AIDDES a CASE Tool. The modeled and stored use cases compose, along with the other activities, except for the coding, the task project. The developer in the role of Driver can edit the use cases while the Navigator can only view them. The model, as well as the other activities, must be finalized by both developers before the task conclusion. Each use case added to the modal generates three requisitions: (1) requisition to produce its description; (2) requisition to build its test cases; and (3) requisition to build its activity diagram.
	Produce Use-case Descriptions - Just like the other descriptions seen in this section, all the model use cases produced by the developers for a determined task must be described in a textual format. Only the developer in the role of the Driver can edit the use-case descriptions – the Navigator can only view it. The activity of describing the use cases must be finalized by both developers as a pre-requirement for the task conclusion.
	Build Test Cases - After generating the requisition to build a test case of a use case, the Driver becomes able to describe its primary, alternative and exception scenarios. Developers in the role of Navigators can only view the test cases produced by the Driver. As a primary requirement for a task conclusion, it's imperative that both members of the team finalize the activity of test cases.
	Build Use-case Activity Diagrams - The Driver must produce an activity diagram for each modeled use case. Developers in the role of Navigators can only view the activity diagrams produced by the Driver. Before the task conclusion, it's also imperative that both members of the team finalize the activity of modeling the activity diagrams.
	Write the Program - Just like the activity of declaration of objectives, the coding activity is done with a synchronous shared editor that demands the revision and acceptance of both members of the team so it can be considered finalized – it's necessary that both members trigger the "Finalize Coding" button. After concluding the coding activity, if there's any new character inserted, its status turns again to "in production". This feature allows that, at any time of the process, the developers can edit the code updating, removing or inserting new instructions or commentaries. Only the developer in the role of the Driver can edit the developed source-code.
	Compile Program - A compiler is called by AIDDES to execute the coded-program compilation – this event occurs only in cases in which the Driver had previously indicated that the program would be written in one of the following languages: C, C++, Java, Fortran 77, ADA 95 or Pascal.
	Save Program - Programs must be saved on disk as basic pre-requirement for their compilation.
	Produce Class Diagram - When producing class diagrams, the developers can describe the class attributes, methods and relations. The developer in the role of Driver can edit the class diagram whereas the Navigator can only view it. The diagram, as well as the other activities, must be finalized by both members before the task conclusion.
	View Activities - The Navigator has read-only access to the activities in a pairing session. The Driver also has read-only access, when s/he connects to the system by himself/herself, without the Navigator.

4. Conclusions And Future Works

Tools that support paired programming have been found in the literature, although it's clear the lack of systems of such nature that allow voice communication between the team members in pairing sessions. In addition, other problems have been detected in available tools, such as: some of them don't allow, internally, the compilation of written programs; none of them has Artificial Intelligence techniques; and none of them seem to support paired projects, tests and reviews.

The present work cared about projecting and producing an intelligent distributed collaborative computer software development environment, named AIDDES, intending to solve all the problems aforementioned. The environment is formed by a synchronous intelligent code editor and a CASE Tool, both shared, in witch the users can accomplish the following activities: (1) declarations of objectives; (2) use cases; (3) use-case descriptions; (4) activity diagrams; (5) test cases; (6) class diagrams; and (7) source-code. It also features voice, Chat and Forum communication during all the pairing session.

The environment is presently in its test phase. We are implanting the system and developing an empirical study about its effectiveness in paired programming process in a System Information course.

It's suggested that, in future works, videoconference communication be implemented as an functionality improvement for AIDDES.

References

Atsuta, S. and Matsuura, S. (2004) "eXtreme Programming support tool in distributed environment". Proceedings of the 28th Annual International Computer Software and Applications Conference, 28-30 Sept., volume: 2, p. 32-33.

Baheti, P., Williams, L., Gehringer, E. and Stotts, D. (2002) "Exploring Pair Programming in Distributed Object-Oriented Team Projects". OOPSLA Educator's Symposium 2002, November.

Baheti, P., Gehringer, E. and Stotts, D. (2002) "Exploring the efficacy of distributed pair programming". Extreme Programming and Agile Methods - XP/Agile Universe 2002, number 2418 in LNCS, p. 208-220.

Bryant, S., Romero, P. and Boulay, B. (2005) "Pair programming and the re-appropriation of individual tools for collaborative programming". In Proceedings of the 2005 international ACM SIGGROUP conference on Supporting Group Work.

Canfora, G., Cimitile, A., Di Lucca, G. A. and Visaggio, C. A. (2006) How distribution affects the success of pair programming. In *International Journal of Software Engineering and Knowledge Engineering*, vol. 16, nº 2, p. 293-313.

Cubranic, D. and Storey, M. A. D. (2005) "Collaboration support for novice team programming". Proceedings of the 2005 international ACM SIGGROUP conference on Supporting group work, November 06-09, Sanibel Island, Florida, USA.

Cubranic, D., Storey, M. A. D. and Ryall, J. (2006) "A Comparison of Communication Technologies to Support Novice Team Programming". ICSE´06, May 20-28, Shangai, China.

Defranco-Tommarello, J. and Deek, F. P. (2005) "An on-line tutorial for collaborative problem solving and software development". In Proceedings of the 6th conference on Information technology education. Newark, NJ, USA, p. 349–352.

Faria, E. S. J. and Yamanaka, K. (2008) "Pair Programming: A Survey (State-of-the-Art in 2007)", [submitted and accepted for publication to the Journal of Brasilian Computer Society].

Faria, E. S. J., Yamanaka, K., Tavares, J. A., Lacerda Pinto, G. H. and Melo, L. H. S. (2008) "Intelligent Software Agents Mediating the Pair Participation in a Distributed Intelligent Pair-Software Development Environment". 3rd IEEE International Workshop on Engineering Semantic Agent Systems (ESAS 2008) in conjunction with COMPSAC 2008. Turku, Finland.

Hanks, B. F. (2002) "Tool Support for Distributed Pair Programming. Workshop on Distributed Pair Programming". Extreme Programming and Agile Methods - XP/Agile Universe 2002.

Hanks, B. F. (2004) "Distributed Pair Programming: An Empirical Study". Extreme Programming and Agile Methods - XP/Agile Universe 2004.

Hanks, B. (2006) "Student attitudes toward pair programming". SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE '06), p. 113-117. [ACM Digital Library]

Hickey, T. J., Langhton, J. and Alterman, R. (2005) Enhancing CS programming lab courses using collaborative editors. In *Journal of Computing Sciences in Colleges*, v.20 n.3, February, p.157-167.

Ho, C., Raha, S., Gehringer, E. and Williams, L. (2004) "Sangam: a distributed pair programming plug-in for Eclipse". OOPSLA workshop on Eclipse Technology Exchange (Eclipse '04), p. 73-77.

Langhton, J. T., Hickey, T. J. and Alterman, R. (2004) "Integrating tools and resources a case study in building educational groupware for collaborative programming". Consortium for Computing Sciences in Colleges, Northeastern Conference.

Mendes, A. J., Gomes, A., Esteves, M., Marcelino, M. J., Bavo, C. and Redondo, M. A. (2005) "Using simulation and collaboration in CS1 and CS2". ITiCSE'05, p. 193-197.

Natsu, H., Favela, J., Morán, A. L., Decouchant, D. and Martinez-Enriquez, A. M. (2003) "Distributed Pair Programming on the Web". In Proceedings of the Fourth Mexican International Conference on Computer Science (ENC'03).

Stotts, D., Williams, L., Nagappan, N., Baheti, P., Jen, D. and Jackson, A. (2003) "Virtual Teaming: Experiments and Experiences with Distributed Pair Programming". Extreme Programming/Agile Universe 2003, p. 129-141.

Visaggio, C. A. (2005) "Empirical Validation of Pair Programming". In Proceedings of the 27th international conference on Software engineering. St. Louis, MO, USA, p. 654-654.

Zin, A. M., Idris, S. and Subramaniam, N. K. (2006) Improving Learning of Programming Through E-Learning by Using Asynchronous Virtual Pair Programming. In *The Turkish Online Journal of Distance Education*, Vol: 7, Issue: 3, p. 162-173.