Collaborative Design using a Shared Object Spaces Infrastructure

Sandro Sawicki[1,2], Lisane Brisolara[1], Leandro S. Indrusiak[1], Ricardo Reis[1]

[1] Universidade Federal do Rio Grande do Sul – UFRGS
Instituto de Informática – Av. Bento Gonçalves, 9500 – Campus do Vale – Bloco IV - Cx.15064
Porto Alegre – RS – Brazil – 9501-970

[2] Universidade Regional do Noroeste do Estado Rio Grande do Sul – UNIJUI
DeTec - Departamento de Tecnologia Rua São Francisco, 501. Cx. Postal 560 98900-000
Ijui - RS – Brazil

{sawicki, lisane, lsi, reis}@inf.ufrgs.br

Abstract

A collaborative design system strongly depends on the chosen collaboration methodology, as well as on its technological infrastructure. This article describes the implementation of collaborative service based on shared object spaces as technological infrastructure and its methodology is based on Pair Programming. This service will be incorporated in a distributed collaborative environment called Cave. The collaboration service implementation presented in this work allows collaboration among designers through a data depository. This service is validated with a diagram editor that is used as case study.

1. Introduction

To handle with the increasing complexity of an IC design, the use of higher abstraction levels is necessary. To deal with these levels of abstraction, tools and methodologies must be built to aid the designer. Usually, these tools have several origins and can be developed for different platform, obliging the designer to constantly change the work environment among distinct hardware/software platforms. It also demands the use of several interface models, increasing the design time [1].

Taking into account such scenario, the Cave [2] environment in under development. The Cave is a new framework for the integration of CAD tools, based mainly in distribution of resources and platform independence. In the Cave2 framework, current version of the Cave environment, it is introduced the support to collaborative design [1, 3].

The need for distributed collaborative design can be easily justified by the following reasons: increasing complexity of designs, demanding of larger teams; shortage of engineers, requiring of manpower in different regions of the world; need for experience sharing among the members of a team.

A collaborative design system strongly depends on the chosen collaboration methodology, as well as on its technological infrastructure used to share design data [4]. These topics will be presented in section 3 and 4, respectively.

This paper presents an implemented approach for collaborative services. The collaboration methodology is based on Pair Programming [5] and the technological infrastructure used for implementing design data repository is based on shared object spaces using Jini/Javaspaces technologies [6, 7]. This collaborative service will be incorporated in the Cave environment and it must be generic to handle all tools included in this environment. A diagram editor, called Blade [8], is used as case study to validate this service.

2. Collaborative Design with Cave2

The collaboration in Cave2 intends to support designers working at the same time and also in different geographical locations. The targets of collaboration among designers are system description building using both textual and visual languages (languages programming, HDLs, block diagrams, schematics and UML diagrams) [3].

The Cave2 collaborative design includes the following features: the design semantic and its graphical/textual representation are modeled by different objects, in order to allow several visualizations – by different designers – from a single design block (separation of concepts); a 2-way update/notify mechanism was implemented, to grant consistency between the design semantic and its graphical representations (update/notify mechanism).

3. Collaboration Methodology

The choice of methodology to support collaborative design in Cave2 is based on reported techniques used in software engineering. The Pair Programming [5] is a

Proceedings of the 15 th Symposium on Integrated Circuits and Systems Design (SBCCI'02)
0-7695-1807-9/02 $17.00 © 2002 IEEE

programming style where two people can work side by side on the same computer, continuously collaborating in the same project, algorithm, code or test. Studies about Pair Programming demonstrate that programming results using this technique increase software productivity and quality.

In the Cave2 context, the Pair Programming methodology was extended to allow collaboration among large groups. This adaptation was proposed in [3], and it was called Paar Programming.

The implementation of this methodology was based on MVC (Model View-Controller) [9] pattern. In this software design pattern the data is separated in three groups: the model, the visualization and the control group. The model represents and stores data. The visualization allows the visualization of data by users, and the control captures the user interactions with data and visualization.

As part of the study of the implementing collaborative work into the Cave environment, two different approaches were described in [3] and are implemented in this environment: *the visually coupled collaboration* and the *visually decoupled* collaboration approach.

In the *visually coupled collaboration* mode, all the designers have the same view of the design data – for instance, all of them see the same diagrams, modules, files, etc. The first case occurs, for instance, when one designer creates a new block of the system, or when a connection between two blocks is removed. The second one, when one designer rearranges the diagrammatic representation of a system, by moving the position of blocks, but without altering its semantics. In both cases, the actions must be notified to all others designers, to ensure the consistency of the collaboration.

In this approach there is only one view, that is stored in a persistence server. In this case, when any data is updated, all designers are notified because every designer has in his/her GUI a reference to that view.

In the *visually decoupled collaboration* mode, the designers do not share the view of the design data – for instance, it is possible to all of them to see different diagrams, but all of the diagrams depict the same model. So, in this case the updates on the view are not broadcasted to other designers, unless it also alters the associated model.

While this second approach allows each designer to organize his/her design workspace, its implementation is harder and several problems must be solved. To guarantee to each user a consistent view in every collaborative session, we started to store also the views for every designer at the end of each session.

4. Technological Infrastructure

Usually, a CAD tool uses complex data types. With an object-oriented paradigm it becomes possible to abstract this complexity by using objects to model these data. Besides, the OO modeling of the design data facilitates the implementation of the collaboration modes proposed in [3]. This section presents a proposal of infrastructure to support objects storage and sharing using a data repository.

The support to an information sharing space, also called as data repository, is a fundamental issue in a cooperative design, as well the technological infrastructure used to implement this repository.

The persistence storage supports cooperative work, serving as an information repository where data can be easily accessed. Essentially, the storage space must support some basic characteristics such as concurrence control, version control, history maintenance of the system objects use, structured and non-structured access, group and its members information management, members interaction management, etc. Theses characteristics are important to grant an efficient and concurrent access to information, allowing notification and knowledge of other group's member activities, follow up of activities evolution, group information, etc.

In [4, 10], some alternatives of data repository technology were researched and three ones were analyzed and compared regarding the implementation of such design repositories: relational database management system (RDBMS), object-oriented database management systems (OODBMS) and shared object spaces.

The following subsection highlights the shared object spaces, that is the technology adopted in this collaborative service implementation.

4.1. Shared Objects Spaces

Following Indrusiak [4, 10] the shared objects spaces provides persistence services without all the complexity of RDBMSs (Relational Database Management Systems) or OODBMSs (Object-oriented Database Management Systems). In order to do that, the query engines - which are the main interface between applications and repositories in RDBMSs and OODBMSs - were substituted by a simpler lookup service. Furthermore, the mechanisms to grant the uniqueness of each data block are not present in the shared object spaces, allowing the storage of multiple copies of a same block.

The data access strategy in shared object spaces also follows the model of read/write transactions, as in the relational model. However, it grants the consistency of the data copies in the applications through an update/notify mechanism: every application is notified if the data they have copied from the repository is updated.

The collaboration service proposal in this work uses Jini/JavaSpaces [6] [7] as technological infrastructure. The Jini technology is a set of APIs organized as a service list and that aid in a distributed systems building. These services are available for clients that want to utilize or combine them to reach a determined objective.

This technology uses several services that can be registered in a table, called the lookup service. In the Jini technology, there is a set of available services that are used in the collaborative service. From this set it can be highlighted: JavaSpaces and transaction services. The first one provides a secure distributed storage system to store an object. The second incorporate the four ACID properties - atomicity, consistency, integrity and durability – preserving the object's security.

4.2. Collaborative Architecture

The object-oriented paradigm and programming based in distributed objects are used in Cave collaborative architecture, as well in the whole framework. This object-oriented approach makes the collaboration service modular, reusable and easily adaptable in the Cave environment.

The collaborative architecture used in Cave2 can be observed in Figure 1. The main parts of this architecture are: tool server, service space, lookup server and clients.

In the service spaces, all services available to collaboration support are initiated. The main available services are persistence, transaction and collaboration service.

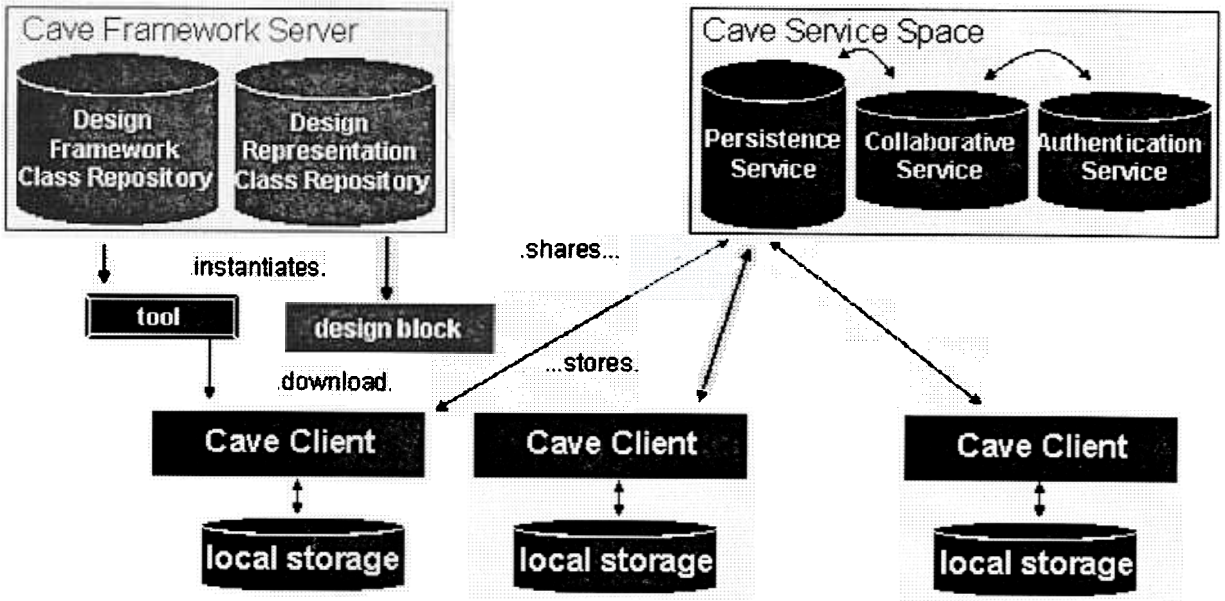

Figure 1: Cave2 Architecture [3]

The persistence server is responsible to store design data, serving as a database and the transaction service treats of the stored design security. The collaboration service works along the others two services, proving the collaboration among designers.

The lookup service is an interface between client and services that contains all references to services. The communication among the tool, the lookup service and the services is transparent for the user. Then, the designer does not need to be concerned about the net configurations or where the designs are stored. The request service can be observed in Figure 2.

The collaboration service is a set of Java classes that allows collaborative work in an application. These classes are added to the application when the service is requested. The service is generic and adaptable to be used by any tools in the Cave environment.

Designers can request a Cave tool through the tool launcher interface. The tool launcher is opened through a socket connection, which can be a HTTP (Hypertext Transfer Protocol) connection created using a web browser. After, the designer must realize his/her authentication and finally, he/she can choose one or more tools available in this environment. When the designer needs to work in a group, he/she can create collaborative sections requesting the use of the collaboration service.

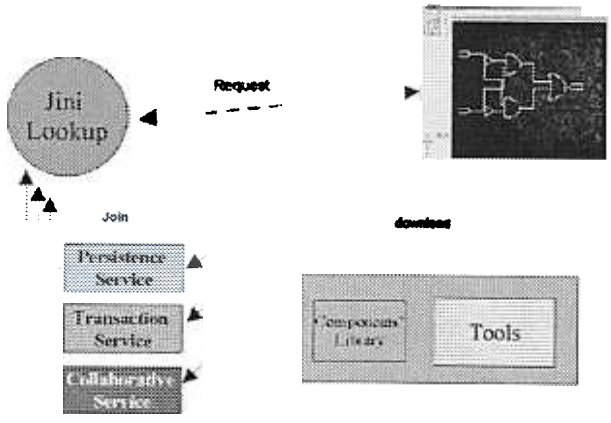

Figure 2: Requesting service

An application of a collaboration service is demonstrated in a case study using a hierarchical diagram editor, called Blade, a tool plugged into Cave.

5. Case Study: The Blade (Block and Diagram Editor)

In [11], it was used the Homero, as case study for collaborative work. This tool is an editor that works with textual descriptions (VHDL, C Language, Verilog, etc). To validate the use of collaborative service with graphical representations, it is used the Blade as second case study.

The Blade is a hierarchical diagram editor target to collaboration. This tool aims to system specification using graphical representation. Although, the current version only allows the construction of logic diagrams, it's structure was modeled to easily be extended to other kinds of diagrams.

In the Blade development it was used an object-oriented approach combined with the use of software design patterns. Moreover, an approach of data separation was used in Blade implementation to allow the implementation of visually coupled and decoupled modes.

In the data separation approach, different objects represent the design semantic and its graphical representation, allowing different views of the same diagram by different designers. The figure 3 shows an illustration for this approach target to Blade.

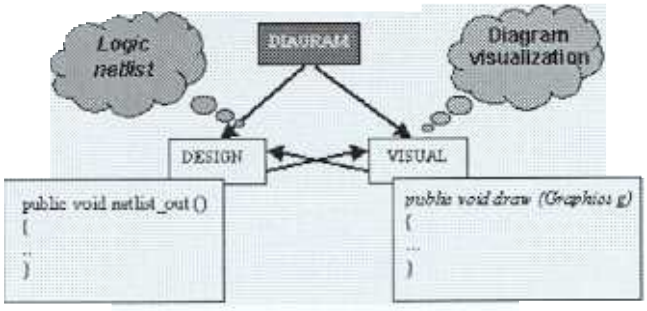

Figure 3: Data Separation approach

In this figure, it can be observed that a diagram representation is formed by two different representations: graphical and semantic. The semantic data, also called design data is used to generate the *netlist* descriptions and the visual data represents the information used to generate diagram views.

The figures 4 and 5 demonstrate the application of visually coupled and decoupled collaborative modes in the Blade tool.

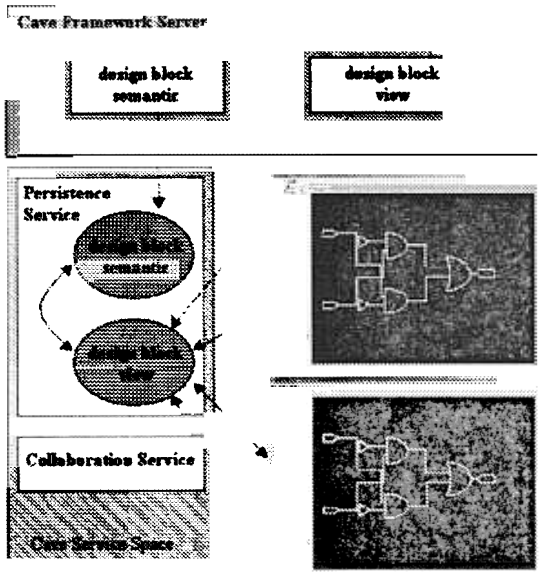

Figure 3: Visually Coupled Mode

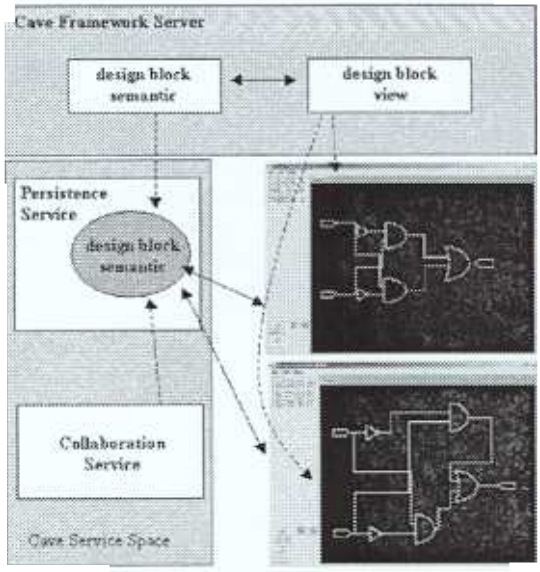

Figure 4: Visually Decoupled Mode

5.1. Collaboration Service Basic Functioning

An important characteristic of electronic system design is modularization. It is used to dominate the design complexity. Complex systems are divided into modules, facilitating the analysis and synthesis tasks by using mo dules with less complexity [12].

A modularization natural consequence is the possibility of working in teams. For example, a microprocessor design can be divided into the design of modules such as ALU, memory, etc. When the specification of the interface among the modules is

ready, different designers or design team can design them independently. However, it is also needed interaction among designers during a design process. When the design of the modules is finished, their assembly can be made.

As described previously, the collaboration methodology used in this work is the Pair Programming. However, this methodology was extended to work with remote machines, allowing collaboration among large groups. In this approach, two or more designers cannot have the write permission at the same time. The designer who has this permission is called *writer*. The others designers can participate as read-only partners, also have known as *listeners*. The listeners can request a write permission to the writer to be able to do changes in the design data.

A collaborative session can work in two different collaborative modes: visually coupled and decoupled. The launch of an update occurs at a different moment in each mode (see section 5.6).

The writer and listeners will be able to interact with every designer through discussions transmitted using a communication tool (chat).

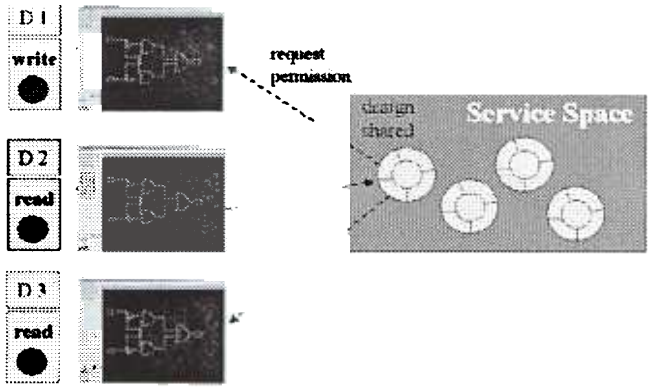

Figure 5: Request of a Write Permission

5.2. Connecting the Space Service

To allow the designer to work in a collaborative mode, he/she needs to establish a unicast connection using a distributed database (service space). This connection provides every collaborative resource to the designers. The connection is made through a LookupLocator protocol in the Lookup Service. The Lookup stored references to persistence (service space), transaction and collaboration services. When the connection is established, the services directly work using the tool.

5.3. Creating Designs

A collaborative design in this implementation is created through the definition of the design name, the responsible person name for this design, the block names, and the names of the designers. Each designer has a password that could be modified by the authorization of the responsible designer.

This operation generates two objects. The first one stores the design and the name of its designers. The second one stores a list of designers that are on-line with the information about their design block and permissions.

5.4. Including Designers

Only the designers included in a design team can participate on collaborative sessions of a design. In the creation mode, the person who is responsible for the design can include designers in the team. Only he/she has a permission to do this operation.

5.5. Transmitting Write Permission

A designer can require the write permission. This solicitation is transmitted to the designer with writer permission that authorize or not to write changes. The permission visualization is possible through a traffic light that is included in the tool interface. It uses the green color to indicate write permission and red color for only read permission.

In this implementation, when a listener designer needs to update the design data, he send a request to the writer designer that can attend or not to his/her request. It is a simple approach, however limited. Other forms to implement the change of permissions is being studied. When the writer designer is gone off, the write permission can be given to the next designer in a requiring list. The other interesting feature is to allow the election of writer designer. In this case, the team designers can vote in a designer to be a new writer.

5.6. Saving Designs

Only the designer who has the write permission can update the design. An update can be or not notified to each team designer, depending on the collaboration mode that is used. The agent that observes the service space launches the notification of an data update. This agent has a different behavior to each collaboration mode: visually coupled and visually decoupled mode.

In the visually coupled mode, when any design block update is made, either in the semantic or in the visualization, every on-line designer receives a notification and the design block update is visualized in the screen. This process can be observed in figure 5. In visually decoupled mode, the designers can have different views of a same functional block. In this case, only changes in the semantic data will be observed by others designers.

5.7. Opening Designs

A designer can open an existing design using two different forms. The first one is searching by designer's name. This search shows the design names in which the designer is participating, as well as the name of others designers that participate in each one of these designs. The second strategy uses the design's name and the design block's name. In this case, a designer enters the information followed by the password in order to control the design access.

6. Conclusions

This paper presents an approach based on shared object space that provides persistence service without all the complexity of RDBMSs or OODBMSs. This approach was developed using Jini/JavaSpaces technology.

The use of Jini/JavaSpaces technology can perform dynamicly the communication among the designers. The database can be refereed in several lookup tables, becoming a typically distributed system. The use of distributed databases facilitates the design data access, turn into flexible, fault tolerant and transparent to users.

Two collaborative modes were proposed in [3] to support collaboration among designers: visually decoupled and visually coupled modes. These collaborative modes were validated by the collaboration service implemented and tested with the Blade diagram editor.

The implementation of the design data repository presented in this paper shows the advantages of using Jini/Javaspace technology. Besides, it proves the advantages of sharing data among designers to support collaborative work.

7. References

[1] Sawicki, Sandro; Projeto Cooperativo no Ambiente Cave; XIII Workshop Iberchip; Guadalajara-México; 2002.

[2] Indrusiak, L. S.; Reis, R. A. L. Ambiente de Apoio ao projeto de Circuitos Integrados baseado na *World Wide Web*. Dissertação de Mestrado. Porto Alegre: CPGC – UFRGS. 19998.

[3] Indrusiak, L. S.; Becker, J.; Glesner, M.; Reis, R. A. L.; Distributed Collaborative design over Cave2 Framework, International Conference on Very Large Integrated (VLSI), Montpellier, 2001.

[4] Indrusiak, L. S.; Glesner, M.; Reis, R.; Comparative Analysis and Application of Data Repository Infrastructure for Collaboration-enabled Distributed Design Environments. In: Design Automation and Test in Europe, Paris, 2002. Los Alamitos: IEEE Computer Society, 2002 (to appear).

[5] Williams, L; Kessler, R.R. All I Really Need to Know about Pair Programming I Learned In Kindergarten. Comm. ACM, 43 (5), p. 108-114, 2000.

[6] E. Freeman, S. Hupfer and K. Arnold. JavaSpaces: principles, patterns, and pratice. Java Series. Reading: Addison Wesley, 1999.

[7] Edwards, W. Keith. Core Jini. Upper Saddle River. Prentice Hall Ptr, 1999. 772p.

[8] Brisolara, L.; Indrusiak L. S.; Reis, R. A. L.; Modelagem Orientada a Objetos de primitivas de projeto de sistemas eletrônicos voltada para colaboração. In: XIII Workshop Iberchip, Guadalajara, México, 2002.

[9] Gamma, Erich; et al. ; Padrões de Projeto. Soluções Reutilizáveis de Software Orientado a Objetos. Trad. Luiz M. Salgado. Porto Alegre, 2000.

[10] Indrusiak, L. S.; Glesner, M.; Reis, R.; Alternatives on Data Repository Infrastructure for Collaboration-enabled Distributed Design Frameworks. (Submitted to SBCCI 2002).

[11] Hernandez, E.; Sawicki, S.; Indrusiak, L.; Reis, R. Homero – Um Editor VHDL Cooperativo via Web. In: VII Workshop Iberchip, Montevideo, Uruguay, March, 2001.

[12] Wagner, Flavio Rech; Ambientes de Projeto de Sistemas Eletrônicos. In: IX Escola de Computação , 24 a 31 de julho de 1994, Recife, Brasil.