Comparative Analysis and Application of Data Repository Infrastructure for Collaboration-enabled Distributed Design Environments

Leandro Soares Indrusiak [1,2], Manfred Glesner [2], Ricardo Reis [1]
[1] *Informática, UFRGS, Porto Alegre, Brazil* [2] *Microelectronic Systems, TU Darmstadt, Germany*
E-mail: <lsi,glesner>@mes.tu-darmstadt.de, reis@inf.ufrgs.br

Abstract

A collaborative design system depends heavily on the chosen collaboration methodology, as well as on its technological infra-structure. This paper presents three data repository technologies and discusses their pros and cons on the role of supporting a collaborative design system.

1. Introduction

The interoperability between design tools has been one of the most important research topics on the EDA field in the last thirty years. Recently, the interoperability between designers started to get attention: collaborative design. There is a need for techniques tailored to ease the communication, coordination and data sharing in groups of designers. The topic discussed in this paper covers technical issues that arise on the design and implementation of the infrastructure to support synchronous collaborative design on distributed environments, more specifically the data representation strategy and the data repository architecture.

2. Infrastructure for data sharing and storage

In order to support people interoperability, the implementation of a data repository must be constrained by the needs of data sharing, concurrent data access, support for multiple views, etc. Furthermore, the implementation should be flexible enough to support different collaboration methodologies, and even support the addition of collaboration methodologies after the deployment of the system (e.g. when the data repository is already populated by design data).

We analyzed three technologies regarding the implementation of such design repositories: relational database management systems (RDBMS), object-oriented database management systems (OODBMS) and shared object spaces.

While the relational model can be suitable for regular enterprise systems, it lacks several of the features which are necessary to support design automation. Its tuple-based approach makes it difficult to model more complex data types. The data access in relational databases is based on read/write transactions, so all the tools using the data from the repository must work with local copies. To insure the consistency between the copies within the tool and the original data in the repository, complicated procedures may be necessary, specially in the case of multi-user collaboration over the same data block.

OODBMS can be divided on two classes, regarding the data access strategy. The first strategy follows the relational model, so the applications work with local copies of the data from the repository. This is mainly because such OODBMSs were built over RDBMSs, or they are actually RDBMSs hidden behind a object-relational mapping interface. The same advantages and disadvantages found in RDBMS apply. A possible exception may be the performance, which can be lower because of the object-relational mapping overhead. The second data access strategy is defined by the concept of single instances of data blocks [1]. It means that the applications don't have local copies of the data stored in the repository, but they have references to the actual stored object. When the data is needed by the application, a remote method call is done to the object, which returns the data. This approach has some advantages on implementing synchronous multi-user collaboration, but in other hand creates a strong dependency on the reliability of the connection between the design tool and the repository server.

Regarding the data modeling, both strategies follow the basic concepts of the object-oriented paradigm, which offers rich semantics to model complex data types. However, some of the OODBMSs require special features from the objects that are going to be stored, such as the use of specific superclasses and the explicit declaration of methods that alter the object state.

Shared object spaces can provide persistence services without all the complexity of RDBMSs or OODBMSs [2]. In order to do that, the query engines - which are the main interface between application and repository in RDBMSs and OODBMSs - were substituted by simpler lookup services. 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 the same block. The data access strategy in shared object spaces also follows the model of read/write transactions, as in the relational model. Although, it grants the consistency of the data copies in the applications through a update/notify mechanism: every application is notified if the data they have copied from the repository is updated. Using a different approach, shared object spaces can also be successfully used as a support for a collaborative design environment. While not allowing direct collaboration over a single data block, this approach can be easily used to implement design data versioning (asynchronous collaboration) or event-based synchronous collaboration. Furthermore, it can be used to support some synchronous collaboration methodologies, where all except one of the members of the collaboration group have read-only access to the data block.

References

[1] G. Mueller and F. Braeutigam. The Ozone Database Project. http://www.ozone-db.org
[2] E. Freeman, S. Hupfer and K. Arnold. JavaSpaces: principles, patterns, and practice. Java Series. Reading:Addison Wesley, 1999.