CUSTOMIZABLE DATA DISTRIBUTION FOR SYNCHRONOUS GROUPWARE

Stephan Lukosch
University of Hagen
Department for Computer Science
58084 Hagen, Germany

Keywords: Collaborative applications, development support, data distribution.

Abstract: The state of a groupware application must be shared to support interactions between collaborating users. There have been a lot of discussions about the best distribution scheme for the state of a groupware application. Many existing groupware platforms support only one distribution scheme, e.g. a replicated or a central scheme, and apply the selected scheme to the entire application. None of these schemes fits well for every groupware application. Different applications and even single applications have different requirements concerning data distribution. This paper describes DreamObjects, a development platform that simplifies the development of shared data objects. DreamObjects supports a variety of distribution schemes which can be applied per shared data object. Additionally, it offers an interface that developers can use to introduce their own distribution schemes.

1 INTRODUCTION

Synchronous groupware brings together users who are geographically distributed and connected via a network. It encompasses a wide range of applications like collaborative whiteboards, text editors or Web browsers. All these applications have to share data to support interactions between their users.

There have been a lot of discussions about the best distribution scheme for the data objects of a groupware application. Many existing groupware platforms support a replicated distribution scheme or a central one and apply the supported scheme to the entire application. None of these schemes fits well for every groupware application. Different applications or even single applications have different requirements concerning data distribution. Therefore, developers should be able to select the distribution scheme per shared data object.

This paper presents the *DreamObjects* platform (Lukosch, 2003), which substantially simplifies the development of shared data objects. DreamObjects is based on *DreamTeam* (Roth, 2000). DreamTeam is a platform for synchronous collaboration and focusses on the coordination and communication of distributed users. DreamTeam offers a hierarchical class library with groupware specific solutions, e.g. awareness widgets, and provides an infrastructure with special groupware facilities. This, e.g., includes a *session manager* that is responsible for starting, joining, and leaving sessions, and a rendezvous manager that determines the actual network addresses of all team members.

In DreamObjects, developers can select between an asymmetric, a replicated, and different adaptive distribution schemes. Adaptive distribution schemes dynamically change the distribution of a shared data object according to a user's working style or according to the topology of the connecting network. Additionally, DreamObjects offers interfaces and building blocks that permit developers to define their own distribution schemes and integrate them into the platform. Developers can specify the distribution scheme of a shared data object at runtime. Thereby, they can use the same data object with different configurations and adapt the configuration to the runtime needs of an application.

Further benefits of DreamObjects are a flexible and extensible concurrency control mechanism, an adaptable notification service, a flexible latecomer support, and a decentralized persistency service. By applying the object-oriented substitution principle, DreamObjects achieves all these benefits with a maximum of transparency for the developer. After an initial con-

Lukosch S. (2004).
CUSTOMIZABLE DATA DISTRIBUTION FOR SYNCHRONOUS GROUPWARE.
In *Proceedings of the Sixth International Conference on Enterprise Information Systems*, pages 70-77
DOI: 10.5220/0002633900700077
Copyright © SciTePress

figuration, developers can use shared data objects like local objects of a single-user application. Developers do not have to care about data sharing issues.

The next section discusses the requirements concerning data distribution. Section 3 compares these requirements to the state-of-the-art. Section 4 and section 5 introduce the DreamObjects platform and its data distribution mechanisms, followed by conclusions.

2 REQUIREMENTS

Roth and Unger introduce an extensible classification model for synchronous groupware (Roth and Unger, 2000). In contrast to former classification models, e.g. Patterson's taxonomy (Patterson, 1995) or Dewan's generic architecture (Dewan, 1996), this model directly addresses the distribution characteristics of a groupware application. It consists of an application scheme and a distribution scheme. The application scheme strictly separates the user interface of an application from the underlying data and algorithms. To classify existing groupware platforms, Roth and Unger identify different distribution schemes: *central state*, *asymmetric state*, and *replicated state*.

Fig. 1 shows the central state distribution scheme. The dotted frames indicate different sites inside a network. The user interface is replicated to every site. A well-known server manages the shared data objects of an application. The main advantage of this distribution scheme is that it is very simple to ensure data consistency and to handle persistency. However, one has to maintain the server, and the server is a bottleneck for the communication between the participating sites. Slow network connections increase the *response time* of an application. For interactive applications, it is a quite important issue to keep the response time low, as this time is directly perceived by the user and a high response time disturbs the interaction of the user with the application.

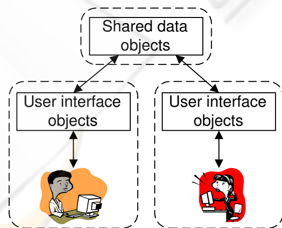

Figure 1: Central state distribution scheme

In the asymmetric state distribution scheme (see fig. 2), an arbitrary participating site fulfills the tasks of a server. Using this scheme, every participant can easily share local data. Apart from this advantage, this scheme has the same drawbacks as the central distribution scheme. However, both schemes, the central and the asymmetric one, can be useful in cooperative applications, e.g. to share large amounts of data or to introduce local data.

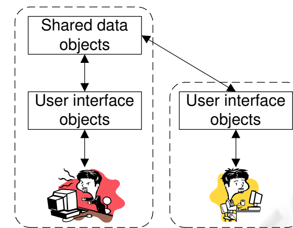

Figure 2: Asymmetric state distribution scheme

In the replicated state distribution scheme (see fig. 3), the shared data objects are distributed to every participating site. As an application can access the shared data objects locally, the response time of an application is reduced. This can be useful in highly interactive cooperative applications, like e.g. cooperative games. However, as already discussed by Molina (Garcia-Molina, 1986), this distribution scheme also has its drawbacks. The runtime system has to use more complex algorithms, e.g. for concurrency control. The network traffic increases to keep the shared data consistent, since every site has to be informed about a modification.

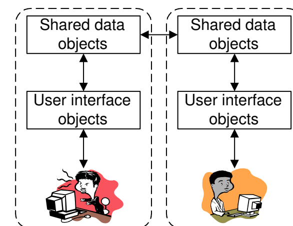

Figure 3: Replicated state distribution scheme

A variant of the replicated distribution scheme is the *partially-replicated* one. In this distribution scheme, the shared data objects are distributed to more than one site, but not necessarily to all sites. Compared to the asymmetric and central distribution scheme, this increases the availability of the shared data. Compared to the replicated distribution scheme, this decreases the network traffic that is necessary to keep the shared state consistent. However, the runtime system still has to use more complex algorithms.

Different approaches exist for a partially-replicated distribution scheme. The shared data objects can, e.g., be replicated to a predefined set of sites. These sites

can access the shared state locally and have to keep the shared state consistent. The other sites choose an arbitrary site to access the shared state. Fig. 4 shows this scenario. On the one hand, this approach increases the availability of the shared data objects, but on the other hand the predefined sites may not be the sites that intensively access them.

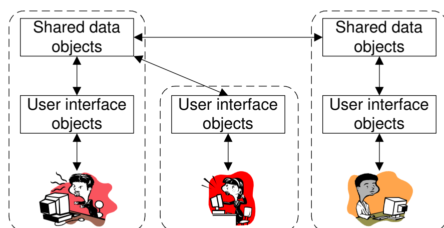

Figure 4: Partially-replicated state distribution scheme

Normally, not all users view or edit all shared data objects. Thus, not every participating site requires all shared data objects at all times. Instead, a site only requires the shared data objects the user currently views or edits.

To take this into account, a platform can replicate a shared data object on demand, e.g. as soon as a user accesses the part of a document that the shared data object represents. When a site does not access a shared data object anymore, e.g. the local user changed his working focus, it has to discard it. Otherwise the set of replicas continually grows and finally the shared data object is replicated to all participating sites. As the working style of a user is not deterministic, this approach can lead to high communication costs.

A partially-replicated distribution scheme that takes the changing working style into account, is called *adaptive* or *dynamic* and was postulated by Gavish et al. (Gavish and Sheng, 1990). Wolfson et al. (Wolfson et al., 1997) describe a distributed algorithm for an adaptive replication that uses a cost-function to adapt the replication scheme of a shared data object. They show that their algorithm compared to a static replication significantly reduces the network traffic.

The discussion shows that every of the presented distribution schemes has its drawbacks and advantages and that none suits well for every groupware application or for all data objects of an application. The asymmetric distribution scheme fits well when a user wants to introduce local data into a collaborative session. The replicated distribution scheme offers high responsiveness. Developers choose an adaptive distribution scheme, if they want to increase the availability of a data object and, compared to the replicated distribution scheme, want to decrease the network traffic. They can also choose an adaptive distribution scheme, if the data of an application can be divided into single logical pieces, e.g. sections or paragraphs in a text document, and they want to adapt the distribution of the data to a user's working style. Thus, to cover all possible requirements of a collaborative application, a platform has to support an asymmetric, a replicated, and an adaptive distribution scheme.

3 RELATED WORK

There exist platforms that use a central, a replicated, or a partially-replicated distribution scheme, others offer a variety of distribution schemes.

Rendezvous (Hill et al., 1994), *Suite* (Dewan and Choudhary, 1992), and *Notification Service Transfer Protocol (NSTP)* (Patterson et al., 1996) are sample groupware platforms, which use a central distribution scheme for the data of a collaborative application. A lot of groupware platforms, e.g., *GroupKit* (Roseman and Greenberg, 1996), *COAST* (Schuckmann et al., 1996), or *DECAF* (Strom et al., 1998) use the replicated distribution scheme. *DÀgora* (Simão et al., 1997) is a sample platform for the partially-replicated distribution scheme.

In contrast to the previous platforms, *TCD* (Anderson et al., 2000), *Clock* (Urnes and Graham, 1999), *GEN* (O'Grady, 1996), and *DistView* (Prakash and Shim, 1994) support a variety of distribution schemes. DistView is a part of the *Collaboratory Builder's Environment (CBE)* (Prakash et al., 1999). Normally, all shared objects are replicated. If this is not possible, e.g., if a shared object accesses a file in the local filesystem, the shared object is maintained at the creating site. Thus, DistView supports a replicated and an asymmetric distribution scheme.

TCD and Clock build groupware applications from components. In both platforms, the components can be customized via a visual programming environment. A developer may, e.g., choose between a central and a replicated distribution strategy. While TCD supports concurrency control for both distribution schemes, Clock supports concurrency control only for the central one.

GEN especially focuses on flexible distribution strategies. It provides shared objects as a high-level abstraction and allows a developer to specify, how a shared object is distributed and how its consistency is maintained. For this purpose, the developer has to be completely aware of the underlying concepts and protocols, as he has to change the implementation. As default, GEN offers implementations for a central and a replicated distribution scheme. Additionally, GEN offers a distribution scheme that is based on migration.

In summary, platforms for almost every distribu-

tion scheme exist. As already argued, every distribution scheme has its advantages and drawbacks. As different applications or even single applications can have different requirements, none of the distribution schemes fits well for every groupware application. For this reason, some of the presented groupware platforms support different distribution schemes, but none supports an adaptive distribution scheme.

4 DREAMOBJECTS

DreamObjects is a platform that simplifies the development of shared data objects. It consists of two parts, an object-oriented framework and a runtime environment. Both are entirely implemented in Java. The object-oriented framework provides building blocks for the development of shared data objects. These building blocks offer a lot of configuration possibilities and even allow developers to integrate their own solutions.

As already mentioned, DreamObjects enhances the DreamTeam platform (Roth, 2000). At our institute, we created several groupware applications with DreamTeam, e.g. a brainstorming tool, a collaborative Web browser, and a distance teaching environment (Lukosch et al., 1999). During the development, we noticed that the major obstacles are concerned with data sharing issues.

In our opinion, the development of a groupware application should be almost as simple as the development of a single-user application. A platform should support a developer such that he can concentrate on application-specific details. For this reason, we extended DreamTeam with DreamObjects. They complement one another, reduce the development costs for a collaborative application, and even allow a developer to reuse existing single-user applications (Lukosch and Roth, 2001).

DreamObjects divides a collaborative application in data objects and user interface objects. The data objects are split up in shared and private objects. The user interface objects control the user interface behavior and display the content of the data objects. Users collaborate by modifying the shared data objects via the user interface of the application.

DreamObjects offers a set of services to manage the shared data objects, e.g. distributed method calls and object creations. Most of the services are handled completely transparent for developers (Lukosch, 2002), i.e. they can use shared data objects like local objects of single-user applications. To achieve this transparency, DreamObjects uses *substitutes* (Lukosch and Unger, 2000), which are based on the substitution principle of object-oriented programming languages. A substitute class extends a developer-defined data class and overwrites some of their methods to add functionality, e.g. the mechanisms for modifying and reading method calls.

As a substitute class offers the same interface as the developer-defined data class, it can easily be used to replace a developer-defined data object. For this purpose, a developer has to call a special registration method of the runtime system, which creates and returns the substitute to the developer. When calling the registration method a developer has to provide the runtime configuration of the shared object. This, e.g., defines how the shared object is kept consistent or how it is distributed to the other participating sites.

When a new shared object is registered, the registering site informs all other participating sites, which results in creating a substitute for the new shared object. Thus, each site can access a shared data object like a local object. However, depending on the chosen distribution scheme only some sites may hold the data of the shared object. These sites are called *data holder*. A data holder can execute a method call locally. A site that does not hold the data of a shared object must involve a data holder in the execution of a method call. Then the substitute for the shared object maps possible method calls to data holding sites. For this purpose, DreamObjects uses mechanisms that reduce the number of involved sites to a minimum (Lukosch, 2002). A reading method call is, e.g., only executed by one data holding site, while a modifying method call is executed by all data holding sites to ensure the consistency of the shared data objects.

To fulfill the requirements as discussed in section 2, DreamObjects offers an *asymmetric*, a *replicated*, and an *adaptive* distribution scheme. When developers register a new shared object, they have to define the distribution scheme for the shared object. Thereby, developers can use the same data class with different distribution schemes.

5 DISTRIBUTION SCHEMES

Let D denote the set of shared data objects used in a collaborative session, let S denote the participating sites, and finally let $DH_d \subseteq S$ denote the sites that are data holders of a shared object $d \in D$.

Each distribution scheme in DreamObjects is controlled by an instance of an `Evaluator` class that must be defined during the registration of a shared object.

5.1 Static Distribution Schemes

DreamObjects offers two predefined evaluators for static distribution schemes. The asymmetric distribution scheme fits well, when a developer wants to allow

the users of a collaborative session to introduce local data, e.g. a video that is stored in the local filesystem of a user. The replicated distribution scheme fits well for highly interactive cooperative application, as it reduces the response time of an application.

DreamObjects uses an evaluator for a static distribution scheme to determine the set DH_d, when a shared data object d is created, and to determine whether a latecomer's site becomes a data holder of a shared object d. Furthermore, the runtime system uses an evaluator to sort the data holders in DH_d when a site $s \notin DH_d$ executes a reading method call. In a first attempt, the runtime system involves the first site in the execution of the reading method call. If an error occurs, the runtime system passes the task to execute the reading method call to the next site. Thereby, it is possible to define arbitrary further static distribution schemes.

If a developer, e.g., develops an application that is used in a predefined network, he can define an specific evaluator and only distribute a shared object to the sites that have a good network connection. For this purpose, a developer has to extend the basic `Evaluator` class and implement the three methods: `getDataHolders`, `becomesDataHolder`, and `sortDataHoldersForRead`. The runtime system calls these methods in the respective cases. Fig. 5 shows the corresponding class hierarchy.

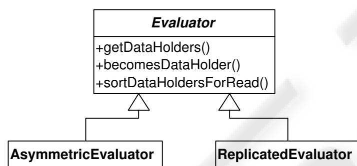

Figure 5: Class hierarchy of static evaluators

5.2 Adaptive Distribution Schemes

In case of a static distribution scheme, the set of data holders only changes when either a site joins the session or leaves the session. In case of an adaptive distribution scheme, the set of data holders can also change in dependence of how a site accesses a shared data object. Developers choose an adaptive distribution scheme, if they want to increase the availability of a data object and, compared to the replicated distribution scheme, want to decrease the network traffic.

To adapt the distribution of a shared object, the runtime system informs the local evaluator for d about the method call, whenever the local site initiated and executed a method call directed to a shared object $d \in D$. It passes the name of the method, a boolean value that indicates, whether the method call modified the content of the object, and the size of the shared data object in bytes to the evaluator. Based on this information and the information about previous method calls, the evaluator can initiate a change in the set of data holders DH_d.

DreamObjects associates a write master $wm_d \in DH_d$ with every shared object. The write master of a shared data object, e.g., ensures that a method call or a method call result is distributed (Lukosch, 2002). When the write master of a shared object leaves the session, DreamObjects deterministically selects a new write master. Concerning adaptive data distribution, the write master of a shared object ensures that the number of data holding sites does not fall below a developer-defined minimum. To ensure the availability of a data object, the specified minimum has to be greater than or equal to two. When $|DH_d|$ falls below the specified minimum, the write master wm_d of the shared object increases the number of data holding sites.

To develop an evaluator for an adaptive distribution scheme, developers have to extend the basic `AdaptiveEvaluator` class (see fig. 6). Compared to the static distribution schemes, following methods must be implemented:

1. `methodCalled`: This method is called by the runtime system, whenever the local site initiated a method call. Based on the passed parameters, the name of the method etc., this method has to decide if the distribution of the shared object has to be changed.
2. `getMinDataHolders`: This method has to return the minimum number of data holding sites.
3. `getNewDataHolders`: When the number of data holding sites falls below the defined minimum, this method has to determine the sites that become a data holder.

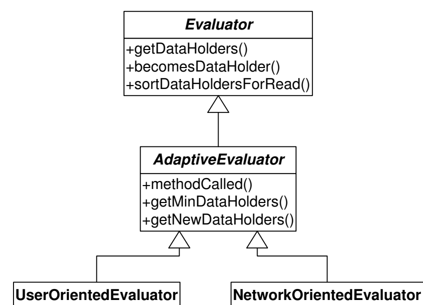

Figure 6: Class hierarchy of adaptive evaluators

DreamObjects offers two pre-defined evaluators for an adaptive data distribution: a user-oriented and a network-oriented evaluator. Both evaluators are based on the following ideas:

1. If a site $s \notin DH_d$ intensively accesses a shared data object $d \in D$, it should become a data holder. So the site can execute reading method calls locally and thus decrease the response time of the application.
2. If a site $s \in DH_d$ does not access a shared data object $d \in D$ frequently, it should give up its role as a data holder. So the number of data holding sites is reduced. This decreases the number of messages that is necessary to keep a shared data object consistent and thus the response time of an application at every site.

5.2.1 User-oriented Adaptive Distribution

Developers choose the user-oriented distribution scheme, if the data of an application can be divided into single logical pieces. For instance, consider a collaborative text editor. A text document is structured into single logical pieces, i.e. words, sentences, paragraphs, sections, and chapters. Depending on the kind of collaboration that a developer wants to support between the users of the collaborative text editor, he can use a shared data object for every single word, sentence, etc. of the text document. Normally, only a few users of a text editor work at the same logical piece of the document. The user-oriented distribution scheme takes this into account and distributes a data object to those sites that intensively access it. As a user's working style can change during a collaborative session, the evaluator reflects these changes in the distribution of the shared data object.

When a new object $d \in D$ using this scheme is created, the evaluator as default selects two of the participating sites as initial data holders. One of the data holding sites is the object creating site s^d, which also becomes the initial write master wm_d. To select the other initial data holders for the shared data object, the evaluator chooses the sites with the fastest network connection.

When a site $s \notin DH_d$ initiates a reading method call, one of the data holding sites has to execute the method call. To determine this site, the evaluator sorts the data holding sites according to their network connection. In a first attempt, a site $s \notin DH_d$ selects the first site in the list to execute the reading method call. If this site fails, the next one is selected, until there is no site left in the list or the method call was executed.

When a site joins a session, the evaluator has to decide whether this site becomes a data holder of d. A latecomer's site only becomes a data holder if $|DH_d|$ is lower than the defined minimum.

To adapt the distribution of a shared data object to the working style of the collaborating users, the evaluator observes how a site accesses and uses a data object. It uses the information about every method call the local site initiates to calculate an *overall load*, a *read load*, and a *write load* in calls per second. The overall load includes all, the read load the reading, and the write load the modifying method calls.

To calculate these values, the evaluator for a data object d at a site s counts how many reading, modifying, and general methods of the data object the local site calls in a developer-defined period of time Δt. Every Δt the evaluator calculates new load values. All load values in calls per second are calculated in the same way. When, e.g., ol_d^s denotes the current value of the overall load at a site s for a data object d, w_{ol_d} denotes the developer-defined weight of the last load value, and mcs denotes the number of all method calls in the last period of time Δt, the evaluator calculates the new overall load $ol_d^{s\prime}$ as follows

$$ol_d^{s\prime} = ol_d^s w_{ol_d} + \frac{mcs}{\Delta t}(1 - w_{ol_d}), \text{ with } 0 \leq w_{ol_d} \leq 1.$$

A developer can specify different weight values and can set minimum and maximum limits for every load. When at a site $s \in DH_d$ a load falls below a minimum limit, the evaluator decides that the site s does not need to be a data holder anymore. Then, the site s informs the other participating sites which remove s from the set of data holders DH_d. After s was removed, it is not notified about modifying method calls anymore and the network load is reduced. To ensure the availability of a shared object, a site may just give up its role as data holder, if $|DH_d|$ is greater than the defined minimum.

When at a site $s \notin DH_d$ a load exceeds a maximum limit, the evaluator calculates the expected transmission time of the shared data object. By specifying a maximum transmission time, a developer can prevent that large data objects are transferred to a site with a poor connection type. If the expected time is below the developer-defined maximum, the evaluator requests the state of the data object, stores it in the local substitute for d, and informs the other participating sites about the new data holder s.

5.2.2 Network-Oriented Adaptive Distribution

Developers use the network-oriented distribution scheme for shared data objects that rarely change and that are mostly accessed with reading method calls, e.g. shared data objects that contain the configuration of a shared application.

To reduce the network traffic and thus the response time of an application in this case, the network-oriented distribution scheme attempts to optimize the distribution of a shared data object with regard to the underlying network structure of the collaborating group. When a new object using this scheme $d \in D$ is created, the evaluator determines a partition of sub-

sets SN_i:

$$S = \bigcup_{i=0}^{n} SN_i, \text{ with } n \in \mathbb{N}$$

For this purpose, the evaluator checks the IP address of every site. Sites whose IP address only varies in the last octet are assumed to be in one subnet and added to one subset SN_i. After this, the evaluator selects one site from every subset SN_i as a data holder of the new shared object. For the subset SN_i with $s^d \in SN_i$, the object creating site s^d is chosen as the data holding site. In the other subnets, the evaluator chooses the site with the fastest network connection. This is done for the following reasons:

- Usually, sites in a subnet have a good network connection to each other, i.e. there is a low network delay and a high transfer rate between the sites. If one site s in a subset SN_i is a data holder of a shared data object $d \in D$, the other sites in the subnet can access d via s.
- As only one site in every subnet becomes a data holder of the shared data object, the number of data holders, compared to, e.g., the replicated distribution scheme, is reduced. This, e.g., decreases the network load for modifying method calls and thus the response time of an application.

When a new shared object is created, the evaluator selects one site from every subnet to become a data holder. Of course, the object creating site becomes a data holder. If the participating sites cannot be divided into as many subnets as the defined minimum of data holding sites, the evaluator sorts the sites according to their network connection. From these sites, the evaluator selects as many sites as are necessary according to the defined minimum.

When a site joins a session, the evaluator has to decide, whether this site becomes a data holder of d. The evaluator checks if the joining site belongs to one of the subsets SN_i. In this case and if $|DH_d|$ is greater than or equal to the defined minimum, a latecomer's site does not become a data holder. If the latecomer's site is the first site of a new subnet, it becomes a data holder in any case.

When a site $s \notin DH_d$ executes a reading method call, it is the task of the evaluator to determine a sorted list of data holding sites. For this purpose, the evaluator sorts the list of data holding sites as follows. For a site $s \in SN_i$, the data holding sites in the same subnet, i.e. $SN_i \cap DH_d$, are put into the first places. After the data holding sites from the same subnet SN_i were added to the list, the evaluator adds the rest of the data holding sites to the list. Again, data holding sites with a better network connection are rated higher than others.

To adapt the distribution, the evaluator again calculates an overall load, a read load, and a write load (see section 5.2.1). A developer can again set minimum and maximum limits for the different load and can specify a maximum transmission time. When at a site $s \notin DH_d$ a load exceeds a maximum limit and the expected transmission time is below the developer-defined maximum time, the evaluator requests the state of the shared object d and the site s becomes a data holder of d.

When at a site $s \in DH_d$ a load falls below a minimum, the evaluator decides that the site s has to give up its role as a data holding site. However, to ensure that each subnet contains at least one data holding site, a site must not give up its role as data holder if there is not another data holder in its subnet. Additionally, a site must not give up its role as data holder if the number of data holders is less than or equal to the defined minimum or if it is the write master.

6 CONCLUSIONS

An ideal platform has to support a variety of distribution schemes, as different applications or even single applications have different requirements concerning data distribution. Compared to the other platforms discussed in section 3, DreamObjects does not only support the common asymmetric and replicated distribution scheme, it also supports two adaptive distribution schemes. One of these distribution schemes adapts the distribution of a shared data object in relation to a user's working style. The other distribution scheme attempts to optimize the distribution of a shared data object with regard to the underlying network structure of the collaborating group. The supported distribution schemes can be applied on a per-object basis and first have to be defined at object registration time. Thus, a developer can use the same object with different distribution schemes. In addition to the supported distribution schemes, DreamObjects offers an interface that enables developers to define their own distribution schemes and to integrate them into the platform.

Nevertheless, there are still open issues. The adaptive distribution schemes use a heuristic approach. A developer can define limits for different load values. Based on these limits, an evaluator decides, whether it changes the distribution of a shared data object or not. As already argued, it is an application- and network-dependent task to choose appropriate limits. However, DreamObjects enables further research in adaptive distribution schemes for collaborative applications, e.g. to evaluate different limits for different application types and network topologies. The results could be used to provide a developer with a set of instructions, how to choose the appropriate evaluator for a data object.

REFERENCES

Anderson, G. E., Graham, T. N., and Wright, T. N. (2000). Dragonfly: Linking Conceptual and Implementation Architectures of Multiuser Interactive Systems. In *Proceedings of the 22nd International Conference on Software Engineering, ICSE 2000*, pages 252–261, Limerick, Ireland.

Dewan, P. (1996). Multiuser Architectures. In *Proceedings of IFIP WG2.7 Working Conference on Engineering for Human-Computer Communication*, pages 247–270.

Dewan, P. and Choudhary, R. (1992). A High-Level and Flexible Framework for Implementing Multiuser Interfaces. *ACM Transactions on Information Systems*, 10(4):345–380.

Garcia-Molina, H. (1986). The Future of Data Replication. In *Proceedings of the IEEE Symposium on Reliability in Distributed Software and Database Systems*, pages 13–19, Los Angeles, CA, USA.

Gavish, B. and Sheng, O. R. L. (1990). Dynamic File Migration in Distributed Computer Systems. *Communications of the ACM*, 33(2):177–189.

Hill, R. D., Brinck, T., Rohall, S. L., Patterson, J. F., and Wilne, W. (1994). The Rendezvous architecture and language for constructing multiuser applications. *ACM Transactions on Computer-Human Interaction*, 1(2):81–125.

Lukosch, S. (2002). Adaptive and Transparent Data Distribution Support for Synchronous Groupware. In *Groupware: Design, Implementation, and Use, 8th International Workshop, CRIWG 2002*, LNCS 2440, pages 255–274, La Serena, Chile. Springer-Verlag Berlin Heidelberg.

Lukosch, S. (2003). *Transparent and Flexible Data Sharing for Synchronous Groupware*. Schriften zu Kooperations- und Mediensystemen - Band 2. JOSEF EUL VERLAG GmbH, Lohmar - Köln.

Lukosch, S. and Roth, J. (2001). Reusing Single-user Applications to Create Multi-user Internet Applications. In *Innovative Internet Computing Systems (I^2CS)*, LNCS 2060, pages 79–90, Ilmenau, Germany. Springer-Verlag Berlin Heidelberg.

Lukosch, S., Roth, J., and Unger, C. (1999). Marrying On-Campus Teaching to Distance Teaching. In *Proceedings of the 19th World Conference on Open Learning and Distance Education*, Vienna, Austria.

Lukosch, S. and Unger, C. (2000). Flexible Management of Shared Groupware Objects. In *Proceedings of the Second International Network Conference (INC 2000)*, pages 209–219, University of Plymouth, United Kingdom.

O'Grady, T. (1996). Flexible Data Sharing in a Groupware Toolkit. Master's thesis, University of Calgary, Department of Computer Science, Calgary, Alberta, Canada.

Patterson, J. F. (1995). A Taxonomy of Architectures for Synchronous Groupware Architectures. *ACM SIGOIS Bulletin Special Issue: Papers of the CSCW'94 Workshops*, 15(3):27–29.

Patterson, J. F., Day, M., and Kucan, J. (1996). Notification Servers for Synchronous Groupware. In *Proceedings of the ACM 1996 Conference on Computer Supported Cooperative Work*, pages 122–129, Boston, Massachusetts, USA.

Prakash, A. and Shim, H. S. (1994). DistView: Support for Building Efficient Collaborative Applications using Replicated Objects. In *Proceedings of the ACM 1994 Conference on Computer Supported Cooperative Work*, pages 153–164, Chapel Hill, NC, USA.

Prakash, A., Shim, H. S., and Lee, J. H. (1999). Data Management Issues and Trade-Offs in CSCW Systems. *IEEE Transactions on Knowledge and Data Engineering*, 11(1):213–227.

Roseman, M. and Greenberg, S. (1996). Building Real-Time Groupware with GroupKit, A Groupware Toolkit. *ACM Transactions on Computer-Human Interaction*, 3(1):66–106.

Roth, J. (2000). 'DreamTeam': A Platform for Synchronous Collaborative Applications. *AI & Society*, 14(1):98–119.

Roth, J. and Unger, C. (2000). An extensible classification model for distribution architectures of synchronous groupware. In *Proceedings of the Fourth International Conference on the Design of Cooperative Systems (COOP2000)*, Sophia Antipolis, France. IOS Press.

Schuckmann, C., Kirchner, L., Schümmer, J., and Haake, J. M. (1996). Designing object-oriented synchronous groupware with COAST. In *Proceedings of the ACM 1996 Conference on Computer Supported Cooperative Work*, pages 30–38, Boston, Massachusetts, USA.

Simão, J., Domingos, H. J., Martins, J. L., and Preguiça, N. (1997). Supporting Synchronous Groupware with Peer Object-Groups. In *Proceedings of the Third USENIX Conference on Object-Oriented Technologies (COOTS)*, Portland, Oregon, USA.

Strom, R., Banavar, G., Miller, K., Prakash, A., and Ward, M. (1998). Concurrency Control and View Notification Algorithms for Collaborative Replicated Objects. *IEEE Transactions on Computers*, 47(4):458–471.

Urnes, T. and Graham, T. N. (1999). Flexibly Mapping Synchronous Groupware Architectures to Distributed Implementations. In *Proceedings of Design, Specification and Implementation of Interactive Systems (DSV-IS'99)*.

Wolfson, O., Jajodia, S., and Huang, Y. (1997). An Adaptive Data Replication Algorithm. *ACM Transactions on Database Systems*, 22(2):255–314.