Notes on structural analysis in a distributed collaboratory

GC van Rooyen and AH Olivier
Department of Civil Engineering, University of Stellenbosch, 7600 Stellenbosch, South Africa
(gcvr@sun.ac.za)

Summary

The worldwide growth of communication networks and associated technologies provide the basic infrastructure for new ways of executing the engineering process. Collaboration amongst team members seperated in time and location is of particular importance. Two broad themes can be recognized in research pertaining to distributed collaboration. One theme focusses on the technical and technological aspects of distributed work, while the other emphasises human aspects thereof. The case of finite element structural analysis in a distributed collaboratory is examined in this paper. An approach is taken which has its roots in human aspects of the structural analysis task. Based on experience of how structural engineers currently approach and execute this task while utilising standard software designed for use on local workstations only, criteria are stated for a software architechture that could support collaborative structural analysis. Aspects of a pilot application and the results of qualitative performance measurements are discussed.

1 Introduction

Technology developments drove the development of engineering applications since the start of the digital computer age, and the requirements of engineering applications in turn drove the development of computer technologies. It may be assumed that this trend will continue. Recent network- and communication technology developments have a severe impact on the structure of engineering applications, since the latter was traditionally developed for single user, single workstation environments. Extending these applications for collaborative use in communication networks demands a major refactoring going to the core of the application structures. Information technology developers and users have to address these issues in the near future. Their success will be determined not only by what technology is capable of, but just as much by creating tools that will encourage team members to collaborate. Towards this, criteria are stated for software tools that could support finite element analysis in a distributed collaboratory, and which also allow the team members to work efficiently. These criteria determine the architecture of the distributed analysis application. The assumption is made throughout that such an application is used by a team of engineers collaborating in the execution of a single project.

2 Criteria for a distributed structural analysis application

Applications for distributed collaboration cannot be successful if the team members using it cannot work efficiently. This places significant demands on how information is exchanged, and how communication between the different parties takes place. Initially, users will measure their efficiency relative to that which they can achieve using standard tools on local workstations. Applications that support finite element analysis on local workstations have been developed over decades and are sophisticated and robust. Engineers that currently use these applications will not be satisfied with anything less than what they have become used to. Based on experience in engineering offices specializing in finite element analysis of structures, specific technical criteria which a supporting technology for distributed structural analysis should meet are listed below. The implications and consequences of the stated criteria are discussed as well.

2.1 Standalone capability

It is ironic that the first criterion of a distributed analysis application is that it should support work in the isolated environment of a local workstation. A collaboratory involving a number of computers and an Internet-based communication infrastructure carries a high risk of failure, or even negligence, in any one of its components. Such an occurrence should not incapacitate the complete environment, i.e. working at unaffected stations should be possible in isolation.

Consequences: The standalone criterion has the following consequences:

(i) There can be no references to remote objects at execution time,

(ii) which in turn implies that objects which are shared in the collaboratory have to be transferred (by value) to the workstations where they are used,

(iii) which implies that the collaboratory should have a reliable project server running a database service which allows team members to check shared objects into, and out of a project database.

Two significant problems stemming from the physical transfer of objects to various locations are the data-volume cost of transfer and the problem of consistency arising from a high degree of redundancy.

2.2 Consistency

Members of the project team make use of information available to them at a certain point in time. If the information base is not consistent, the decisions and actions they take are based on outdated information. Such situations have to be avoided as far as possible, since they have a negative influence on the economics of the project and on the time schedule. Team members share information via the project database. Access to objects checked out of the project database by one team member cannot be locked until they are returned since engineering transactions are typically too long to allow that. As a result changes in the state of objects have to be tracked by versioning. The consistency requirement forces one to address the two issues discussed in the subsections below.

2.2.1 Versioning.

Fullscale versioning: A version of an object is a snapshot of its state at a given time. A fullscale versioning mechanism would record each change to each object, including changes occurring due to dependencies amongst objects. The advantage of fullscale versioning is that the complete development history can in principle be reconstructed with the aid of the versioned information. However, the possible proliferation of versions due to dependencies amongst objects has to be considered.

Dependency: An object y is called directly dependent on an object x if at least one of the following conditions is satisfied: Methods of object x modify attributes of object y, or methods of object y read attributes of object x. (Pahl and Beucke 2000).

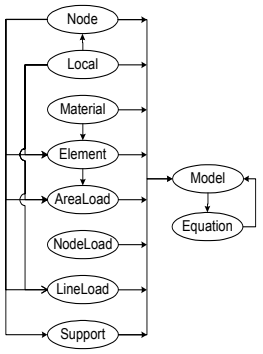


Figure 1: Direct dependency graph

Finite element dependencies: The possible direct dependencies amongst objects of a finite element model are shown as a directed graph in figure 1. Changing a particular object may set off a chain reaction which can be studied by following its effect through the graph. In a more complicated graph the dependencies could be found by computing the transitive closure of the direct dependency relation. A change to a `Local` object, for example, affects the connected `Node`, which in turn affects associated `Element`, `AreaLoad`, `LineLoad` and

`Support` objects. Changes to any of these affects the `Model`, which in turn affects the `Equation`. In a fullscale versioning environment, new versions of the `Local` itself, and of all the dependent objects would be required. In spite of the relatively simple structure of dependencies amongst the analysis objects, a proliferation of versions would result, leading to a complete loss of oversight on the part of the users.

Fundamental change: The modification of certain attributes of an analysis object represent fundamental changes to the object. Changing, for example, the coordinates of a `Node` constitutes a fundamental change to the `Node`, and eventually requires the creation of a new version in order to note the change. A `Node` object also has an array attribute in which the displacements of the node are stored when the results are interpreted. All nodal displacements are contained in the primal vector of the `Equation` as well, and are substituted in the nodes in a simple operation. This creates the possibility that the displacements in the `Node` itself may be regarded as transient information, since the model's primal vector can be stored and used to backsubstitute nodal displacements when required. Exploiting this possibility is convenient, since the same `Node` may then be used in a number of models.

User controlled versioning: In order to avoid the unmanageable growth of versions associated with fullscale versioning, a controlled versioning procedure should be used which takes account of the special properties of the analysis objects and the work pattern associated with the analysis task: The appropriate time to create new versions of fundamentally modified objects depends on the intentions of the engineer. Consequently, new versions, where required, should not be created automatically, or right away. Typically the completion of a model which, in the view of the engineer, yields useful results is the time when the model and its components should be versioned. In this way the number of versions are minimised, and the versioned objects are associated with meaningful models, thereby reducing the loss of oversight significantly

2.2.2 Change propagation.

Service and control: Structural analysis is a technically complex task, and many factors are taken into account when decisions are made. In an analysis collaboratory, the decisions of a member using certain common objects may have a big impact on existing models, and the work of other members. A single change to an object may invalidate the computed results of a model, and recomputation may involve hours of processing time. Consequently strict propagation of changes and control of consistency are not considered to be a suitable mechanisms in a structural analysis collaboratory. Experience has also shown that too much control stifles the collaboration process. (Borghoff 2000) lists a number of concepts which were implemented in the design of a successful product supporting collaboration in a shared information space. Particularly relevant are:

- Overwriting of existing information, and hard consistency requirements are avoided. New entries are simply appended.
- Users recognize conflicting replicas easily and can solve the conflicts manually.

The collaborative structural analysis environment requires a change propagation and consistency service, which

- advises members of the analysis group of modifications made to certain objects (depending on their mode of usage, described below) which they are using, and
- provides the neccessary information to pursue the matter, if required

Mode of usage: The components of finite element models are relatively simple objects, and members of the analysis group know in what way they will be using objects that they check out of the project database. In some cases they would like to ensure that the state of certain objects

is the same in all models where these objects are used. However, it is just as likely that they want to use certain information of objects but want to bring about changes that should not be reflected in other models in which these objects are used.

Consistent mode: A consistent object has the same state in all models in which it is consistent. If a user has the necessary rights to employ an object in consistent mode, and needs to change its state, the changes either have to be reflected in other models in which the object is used, or the versioning graph of the object splits with different versions of the object in the different models.

Free mode: If a user employs an object in free mode, changes to the object only affect the local model and are not propagated to other models where the object is used. This mode is useful in the collaborative environment since two objects may be related without having the same state.

Consistency mechanism: When an object is checked out of the project database in consistent mode, the purpose of the consistency service is to ensure that any fundamental change to the object is reported to other models in which it is used. However, it is considered the duty of the owner of a model to act on reported changes. The affected models may either be in a dormant state in the project database, or in dynamic use (i.e. under construction) at one or more of the workplaces.

- *Dormant models:* The consistent mode evolution of objects is recorded in the project database. When a model which contains previous consistent-mode versions of objects is checked out of the database, the user is informed that later consistent-mode versions of the objects are available.
- *Dynamic models:* Changes to objects which have been checked out of the project database in free mode are not reported to other members using the same objects, and the free-mode evolution of objects is not recorded in the database. However, fundamental changes to objects which have been checked out in consistent mode immediately propagated to other members that are using the same object in consistent mode. Similarly, when a new version of a consistent mode object is created, the new version is propagated to the other consistent mode users of the object. Further action is the responsibility of the member being notified. While an object is checked out, changes to it are accumulated. The accumulated changes are reported to a member who checks out the object at a later point in time.

Consequences: The consistency criterion, applied as described above, has the following implications:

(i) Objects recognize fundamental changes to themselves, they are versioned and the creation of new versions is controlled by the user.

(ii) Objects know their mode of usage and has a method to notify other locations of fundamental changes when used in consistent mode. They also have a method to update their state in reaction to a change notification received from another location.

(iii) The analysis collaboratory provides a versioning service and a consistency service. Both services provide for buffering to allow users to work in standalone mode. It is practical to run these services on the same project server which provides the database service.

2.3 Flexibility

One of the basic rules in the development of engineering software applications is that the aim should be to provide tools, while enforcing as little policy as possible in the use of the application. This enhances the flexibility of the application with respect to the preferences of its users and the special requirements of a given project. For example, the predefinition of models and other structured sets lend a useful structure to the information base. However, as the project evolves, models and sets may have to change, or be removed, and new ones formed, or a member of the analysis group may wish to use the geometric information of a model, but none

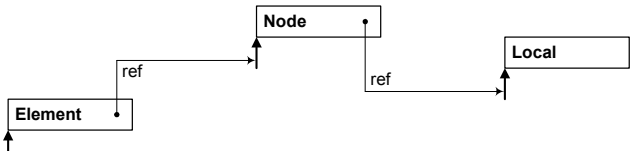
of its boundary conditons. The creative nature of engineering solutions is stifled by applications that forces the user to follow certain policies, and such applications are generally avoided by engineers. Increased flexibility of the distributed analysis application is obtained by leaving much of the management of information up to the users, in the following ways:

- **Free selection of objects:** Selection of objects in the project database is not restricted to any predifined set. In principle, any single object in the project database can be selected individually, and sets can be formed by flexible set-forming relations.
- **User-specified time of updating:** The problem of maintaining a consistent information base is a key issue in distributed environments (Pahl and Beucke 2000). Suitable change propagation algorithms for delayed updating are still under development, and continuous change propagation may not be practical. Furthermore, especially in structural analysis, some alternatives which team members choose to investigate may not be of common interest and should not persist in the project database. Users have to control the updating time of the common information base. This has the additional advantage of providing opportunity to solve compatibility problems between versions before updating the information base.
- **User-specified area of updating:** The effect of certain changes, which may be made frequently, may be localised in the project database. Also, users may want to retain some of the modifications made during a worksession, while discarding others. Users must be able to specify an area of the information base which requires updating.

Consequences: The flexibility criterion places demands on the functionality of the project database management service, as well as on the structure of the distributed analysis application.

(i) *Relational database management service:* In effect, the free selection of objects and the user-specified area of updating described above demands that the database management service should implement a flexible interface. This makes a very strong case for using relational database technology for the project database. The success of this technology has its roots in the extreme simplicity of the relational data model, which requires a single datastructure: tables with rows and columns containing scalar datatypes. The Structured Query Language (ISO-ANSI SQL), which has well defined operations with a strong mathematical basis, provides an extremely flexible interface to the database content, while the programming interfaces ODBC (Open Database Connectivity) and JDBC (Java Database Connectivity) provide SQL-services to program developers. Since the project database is only used for checking objects in and out, the frequency of conversion between the object model of the application and the relational database model is low and should not cause unmanageable performance bottlenecks.

(ii) *Application structure based on persistently identified objects:* A typical action in a collaborative environment would be the case where a user exchanges an object in a model with an object created by another team member, or where a model is constructed by combining components from related models that were constructed earlier. The criterion to allow free selection of objects supports these actions. On the application side, however, the actions described above may be problematic to deal with. Consider the following example: Three components of a finite element model in computer memory are shown in the diagram below.



An Element holds references to its Nodes in order to retrieve information like coordinate positions from them. There may be a Local coordinate system at a Node, in which case the Node will hold a reference to its Local system, from which information regarding the coordinate transformation can be retrieved. If one considers the case where any one of these components has to exchanged with one received from a collaborating source (e.g. the Node), it is problematic to re-establish the references to and from the exchanged component. This problem can be solved in an elegant way: Each component (object) is assigned a String identifier which is unique within the scope of the project in which it is used. The application maintains an objectmap which maps the persistent identifier to the object reference. All components know the persistent identifiers of other components they depend on, and use these to obtain the references when required at runtime (van Rooyen 2002). The analysis collaboratory provides a naming service capable of issuing meaningful names to project objects. This service provide for buffering to allow users to work in standalone mode. It is practical to run this service on the same project server which provides the database service, the versioning service and the consistency service.

2.4 Media suitability

Recent advances in computer networks have connected the computer to a multitude of conventional and novel media, not all of which are practically useful in the engineering process. However, two types of media cannot be ignored, namely support of verbal communication between members of a collaboratory, and support of mobile devices.

Verbal communication: A significant part of a project's information, particularly the interdependence between different pieces of information, is person-based and is conventionally exchanged by speaking. Apart from information exchange, another important goal of the speech act is the requesting and fullfilling of commitments between the conversation participants (Turk 2000) (Fruchter et al. 2000). In distributed environments, however, the danger of ambiguity about what exactly is being discussed is real. Supporting participants in a conversation to simultaneously look at the same drawing, and to point at objects in the drawing while conversing about issues of importance, reduce the possibility of misunderstanding significantly.

Mobile devices: Handheld computers connected to wireless networks present special problems as a result of their size and the speed and reliability of wireless communication. Complete models comprising of large data volumes cannot be transmitted or viewed and processed. The information structure has to allow for the selection of small units of information according to different engineering semantics based criteria.

Consequences: As far as the structure of information in the collaboratory is concerned, the provision of communication support and the use of mobile devices do not have implications over and above those already demanded by the criteria discussed before. Support of pointing at objects in a drawing displayed at various locations is a relatively simple implementation issue.

2.5 Navigability

In a collaboratory, team members are constantly looking for and exchanging information. It is therefore essential that members are able to find information as easily and quickly as possible. A known problem of solutions attempting to address the navigation problem is that pre-programmed conventions regarding search criteria may not be suitable for users of such a system at all. A flexible navigation system is required.

Consequences: Information is exchanged via the project database, and the flexibility of the database management service, provided by the Structured Query Language (SQL), has already been discussed. The full power of SQL can be made available at the user interface of the distributed analysis application as a matter of implementation.

2.6 History of decisionmaking

The development of a project depends on decisions taken as events unfold. A decision taken at a certain point in time should take relevant earlier decisions into account. A reconstructible history of decisions affecting the project, which also captures the intent behind the decisions, will enhance the project members' understanding of the reasons behind changes, reduce misunderstandings, facilitate backtracking, and provide an overview of the project history.

Consequences: Versioned objects and free- and consistent mode usage of objects have already been introduced. The version is available as a holder for the reason why a change was made, i.e. why a new version was needed, and consistent mode changes are typically more important than free mode changes. Additional concepts and services are not necessary to record the history of decisionmaking, and providing for it is an implementation issue.

2.7 Responsiveness of the distributed application

A large number of applications treat structural analysis in a local environment, and these applications are widely used. The responsiveness of these applications has received much attention from their developers. The structural analysis task is characterised by a modelling phase, i.e. when an analysis model is being constructed, an algorithmic phase, i.e. when the solution algorithms are being executed, and an interpretation phase, i.e. when the results of the analysis are being interpreted. A latency (response delay), which is proportional to the number of unknowns being solved for, is associated with the algorithmic phase. This is accepted by users as being unavoidable. However, during modelling and interpretation of results any lack of responsiveness is experienced negatively by users and has to be avoided.

Consequences: In the analysis collaboratory objects are exchanged amongst members of the analysis group in an effort to minimize the total amount of work and enhancing the integrity of the results. The highest volume of exchange takes place during the modelling phase, none during the algorithmic phase, and some exchange takes place during the interpretation phase. The physical volume of data being transferred as a result of collaborative work has to be minimized. Furthermore, implementation efforts have to be made to sustain the responsiveness of the application during the modelling phase.

3 Structure of a distributed analysis application

The structure of the distributed analysis application is determined by the criteria described in section 3. The conceptual structure is summarized in figures 2 and 3. Implementation aspects are described in (Olivier and van Rooyen 2004).

Distribution of objects: The members of the analysis group check objects in and out of the project analysis database, subject to access control. The objects are persistently identified and versioned, and are singly identifiable and retrievable from the database. Structures built into the database allow for the selection of models, as well as the respective sets of components of any model. An object may be checked out in free mode or consistent mode. In the latter mode a consistency service is required.

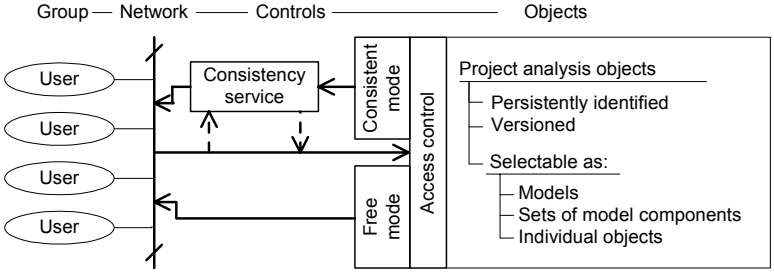


Figure 2: Distribution of objects

Distribution of methods: In the analysis collaboratory, members of the analysis group create, process and interpret the results of finite element models comprising of objects which are shared amongst the members. The methods of the distributed analysis application are executed on the members' local workstations, operating on objects residing in the local runtime memory of the workstation.

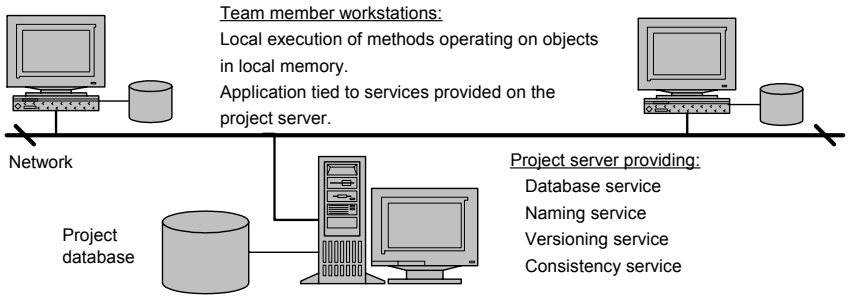


Figure 3: Distribution of methods

To support collaborative sharing of objects, the objects need to be:

- persistently identified with the aid of the naming service,
- versioned with the aid of the versioning service,
- checked in and out of the project database and distributed from there to other members of the analysis group with the aid of the project database service,
- usable in free mode or consistent mode, the latter case being supported by the consistency service.

The above mentioned collaboratory services are distributed applications with functionality at the workplaces supported by methods on the project server. The application maintains structures whereby runtime connections between objects (i.e. references) can be established on the basis of the persistent identification of objects.

4 Overhead cost of distributed work

Members of an analysis group experience the impact of working in a distributed collaboratory by executing the following collaborative actions:

- Checking objects into the project database.
- Checking objects out of the project database.
- Versioning objects that are created or modified.
- Obtaining persistent identifiers of objects that are created.
- Processing changes propagated through the consistency service.

Processing overhead: The additional operations required by the above actions need execution time on both the project server and the local workstation, and a volume of data associated with each action has to be transferred over the network. Using a pilot distributed analysis application structured as described in section 4, the sources of latencies due to these actions, and a measure of their magnitudes, were investigated in (van Rooyen 2002). Two specific results were obtained:

(i) Latency due to additional processing at the workstations is negligible.

(ii) Latency due to transfer of information can be significant. This problem can be alleviated in two ways:

- **Minimization of data volumes:** Objects are transported over the network in their streamed format. The standard streamed format of objects is not necessarily optimized with respect to volume, especially if datastructures like arrays and lists are involved. The programmer has to take control of the conversion to streamed format in order to minimize the volume. Techniques to achieve this are described in (Li 1998).
- **Buffering and sub-processing:** For all of the collaborative actions listed above the use of buffering and the spawning of sub-processes can significantly alleviate the problem of latency due to network transfer delays. Checking objects into the project database can be completely taken care of in a sub-process, and persistent identifiers and new version numbers can be buffered on the local workstation so that they are instantly available. The action of checking objects out of the project database can also be delegated to a sub-process, allowing other work to be performed while the objects are being downloaded from the project server. However, if the objects are required before work can proceed, the latency due to downloading them cannot be avoided.

5 Conclusion

The structure of a distributed structural analysis application which can support an analysis collaboratory was described. The structure is based on user criteria that are judged to enhance collaboration by supporting team members to work efficiently. Four collaboratory services are required, namely a project database service, a versioning service, a naming service and a consistency service. The application uses the collaboratory services to provide the collaborative actions of exchanging objects and reacting to change notifications.The application has to be able to establish reference-connections between objects based on the fact that the objects are persistently identified, and uses the naming service to obtain the persistent identifiers of new objects. The objects are versioned, aware of their mode of usage and whether they have undergone a fundamental change, capable of generating change notifications and reacting to incoming change notifications, and they can generate a compact streamed format of themselves. Qualitative tests indicate that acceptable performance levels can be achieved by an application which is structured as described.

6 References

Borghoff, U.M., Schlichter, J.H., Computer-Supported Cooperative Work, Introduction to Distributed Applications, Springer-Verlag, 2000, ISBN 3-540-66984-1.

Fruchter, R. et al, *To See or Not to See: The Role of Visibility and Awareness in Videocconference-Mediated Teamwork*. In Fruchter, R, Ed., Proc. 8th Int. Conf. on Computing In Civil and Building Engineering, p.852-859, Stanford, California, Aug. 14-16 2000, ASCE.

Li, Liwu, Java Data Structures and Programming, ISBN 3-540-63763-X, Springer-Verlag, 1998

Olivier, A.H. and van Rooyen, G.C. *An application-centred framework for distributed engineering applications.* Paper submitted for the 10th Int. Conf. on Computing In Civil and Building Engineering, Weimar, Germany, 3-5 June 2004.

Pahl, P.J., and Beucke, K, *Neuere Konzepte des CAD im Bauwesen: Stand und Entwicklungen*, Keynote Vortrag zum IKM2000, Bauhaus-Universität Weimar, 22-24 Juni 2000.

Turk, Z.,*Communication Workflow Approach to Computer Integrated Construction (CIC)*. In Fruchter, R, Ed., Proc. 8th Int. Conf. on Computing In Civil and Building Engineering, p.1094-1101, Stanford, California, Aug. 14-16 2000, ASCE.

van Rooyen, G.C., *Structural analysis in a distributed collaboratory*, Dissertation, University of Stellenbosch, Department of Civil Engineering, Stellenbosch, 2002.