ACCESSING FINITE ELEMENT ANALYSIS RESULTS THROUGH AN EXTENSIBLE OBJECT-ORIENTED QUERY LANGUAGE

CC99/1999/000181

K. Orsborn
**Department of Civil and Environmental Engineering,
Massachusetts Institute of Technology, Cambridge, United States of America and
Department of Computer and Information Science, Linköping University, Linköping, Sweden**

(Presented at The Seventh International Conference on Civil and Structural Engineering (CC99), Oxford, United Kingdom, 13-15 September 1999)

Abstract

Database technology will play an important role in providing an efficient management of the information resources in future computational environments in science and engineering. Its powerful capabilities for data management can provide great leverage to the implementation of complete engineering information systems as well as for single scientific and engineering applications.

This paper presents how modern extensible database technology can be utilized within engineering software to provide a query-based interface for accessing and evaluating numerical results from finite element analysis computations. By integrating database technology, the application will be equipped with and can take advantage of general database mechanisms making all data immediately available at the query-language level.

Examples show how the abilities to access and compose data increase dramatically by the introduction of domain models, i.e. a high-level query language-based conceptualization and representation of application-specific concepts and operations.

1 Database Technology as a Foundation for Engineering Software

Database technology is more and more becoming an integral part of *engineering information system* (EIS) environments. Its role as a key enabling technology for managing product data is already established, Cattell [1], but the use of general database technology within computationally intensive applications is not yet well investigated. However, earlier work has shown how modern database technology can be integrated with engineering software, such as software for *finite element analysis* (FEA) [2, 3, 4, 5] and *multibody system* (MBS) *analysis* [6], to support general database capabilities. This paper will present how to provide a query-based interface for accessing and evaluating numerical results from finite element analysis computations.

The earlier work on the FEAMOS [2, 3, 4, 5] system has shown how the AMOS II [7, 8] *extensible* and *main-memory resident database management system* (DBMS) can be integrated with the TRINITAS [9, 10] FEA program. Hence, by embedding the DBMS within the FEA application, the application can take immediate advantage of generic database facilities, such as a *storage manager, data model, metadata, query language, query processing, transactions*, and *remote access* to data sources. Furthermore, true extensibility makes it possible to handle *tailored data representations* and *operators* that the application requires.

In AMOS II, new data representations and operations can be seamlessly integrated and made available in the AMOSQL query language [7]. This means that complex and composite analysis operations can be performed explicitly in a database query without any influential performance degradation [4]. Database mechanisms, such as indexing, can even help make computations more efficient without requiring specialized implementations.

The result evaluation activity includes calculation and interpretation of various physical quantities, such as stresses, displacements, and eigenvalues that is evaluated according to some design criteria. In a conventional FEA application the evaluation process is supported through “hard-wired” and compiled program procedures that restricts the set of possible options the engineer has at his hands. The evaluation process is usually an interactive process where computer graphics plays an important role. For instance, the engineer can choose to evaluate stress and displacement fields by displaying single or multiple values from the analysis and displaying the results in a graph, as an iso-contour plot or another of the available display options. Hence, normally the engineer evaluates an analysis by making various menu-based selections that activate the appropriate program routines that display the result.

In our approach, all analysis data is immediately made accessible through the query language. Since all data are available at the query language level, the user has numerous capabilities to compose queries for *retrieving, combining, constraining* and *transforming* data.

The paper will show how to associate, aggregate, compose, and constrain data for retrieving single and multiple results. It will also be shown how derived functions can be used to filter data in the retrieval process. Furthermore, it will be shown how tailored data structures can be used, how user defined operations can transparently extend the query language and how to perform composite analysis operations explicitly in a database query. Ideas how to combine the query interface with graphical capabilities will also be presented.

2 Database Management Systems and Query Languages

Database technology has traditionally been used primarily for administrative applications. However, through the development of modern technology that supports richer and extensible data models, Stonebraker [13], such as object-oriented data models, database management systems (DBMSs) have started to attract more interest from the technical engineering community.

Current standardization efforts like SQL:1999 by ISO/ANSI [11] and ODMG 2.0 (including the OQL query language) from the Object Database Management Group (ODMG) [12] that will provide standards to support object-orientation in database languages will hopefully bring a more solid ground for developing scientific and engineering database applications.

AMOS II [7, 8] is a *light-weight* and *main-memory resident* DBMS. It is based on an *extensible* object-oriented data model that include the three basic constructs *object*, *type*, and *function*. AMOS II also include a complete declarative, object-oriented and extensible query language, AMOSQL [7], used for defining, querying, and updating the database. The AMOS II can be classified as an *object-relational* DBMS according to Stonebraker and Moore [13] but it is not an extended relational DBMS since it is based on an extensible object-oriented data model.

AMOS II is further designed to act as a mediator [8, 14] software for mediating data between a heterogeneous set of data sources and applications in a distributed computing environment.

3 Query Access to Finite Element Analysis Results

In the FEAMOS system [4, 5] was shown how the AMOS II DBMS could be integrated with the TRINITAS FEA program. The embedding of the DBMS equips the FEA application with the functionality of a general-purpose DBMS. This means that the FEA application can take immediate advantage of these facilities. Furthermore, the extensibility of the DBMS also allows the introduction of application-specific types and operators as well as tailored data structures and algorithms. All types of extensions can be transparently integrated in the query language and handled by the query processor. The query processor can also be extended to take advantage of information required to be able optimize queries that include this extended functionality, such as cost measures, new types of indexes, and optimization algorithms.

The query language can be accessed programmatically from within the application or interactively through a database browser or correspondingly.

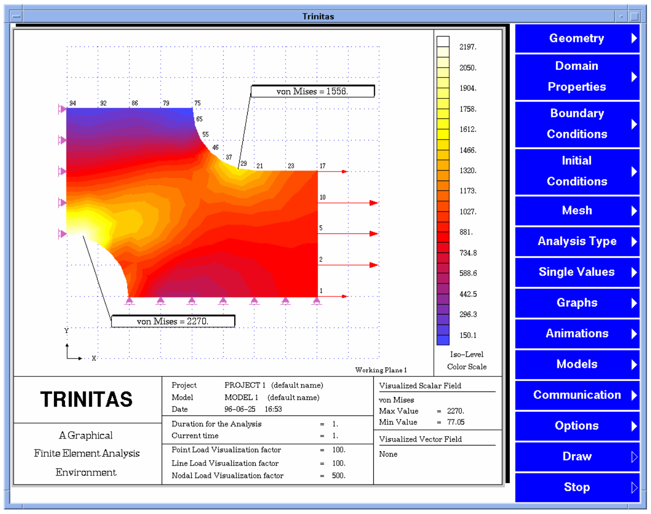
Programmatically, the query language is accessed through an application programming interface (API) by calling precompiled database functions or by evaluating complete database queries. This functionality can be applied both locally in the same process and globally between remote processes. Currently, their are interfaces implemented that uses Java, C, or LISP. Actually, the interface capabilities exceeds that of a normal API since it, for instance, can handle and optimize multi-database queries [15, 16]. Hence, the query processor can handle and optimize access to multiple and heterogeneous data sources.

This section will present how data can be accessed, through a database query language, applying two different approaches for integrating the database. The difference between the two approaches are dependent on at which level the database integration has been performed. First, it is shown how an integration can be done at a fairly low level where a database array representation replaces an application-specific representation. A second approach is then presented where it is shown how additional capabilities can be gained by introducing a conceptualization at a higher level.

Examples shown later in this paper will show how the AMOSQL language can be used interactively by posing queries to the database and retrieving data from the database. It will also be shown how expressions for data and operator modelling can be defined.

3.1 Conventional FEA Result Evaluation

The result evaluation activity includes calculation and interpretation of various physical quantities, such as stresses, displacements, and eigenvalues that should be evaluated according to some design criteria. As illustrated in Figure 1, TRINITAS supports this evaluation graphically by facilities for displaying, for instance, stress and displacement fields as either single values or multiple values as graphs or contour plots. The calculated results are stored in main-memory arrays that are accessed by a set of built-in FORTRAN procedures. By making various menu-based selections, the user can activate appropriate routines and display the result.



Project PROJECT 1 (default name)
Model MODEL 1 (default name)
Date 96–06–25 16:53

Duration for the Analysis = 1.
Current time = 1.

Point Load Visualization factor = 100.
Line Load Visualization factor = 100.
Nodal Load Visualization factor = 500.

Visualized Scalar Field
von Mises
Max Value = 2270.
Min Value = 77.05

Visualized Vector Field
None

Figure 1. *An example that shows how analysis results can be presented to the user in TRINITAS. The picture includes coloured iso-levels of von Mises stresses, the displacement field, and two single stress values.*

3.2 FEA Data Access Through Queries

The provision of a DBMS and a query language provides general database capabilities to the application including a storage manager, data model, metadata, query language, query processing, transactions, and remote access to data sources. The query language makes it possible to retrieve and manipulate data in a declarative manner that relieves the user from specifying the procedure for how to access data.

DBMSs that permits the use object-oriented data modelling usually permits the user to extend the system with user-defined types and operations. Advanced systems also handles optimization of queries including this type of user-defined information.

With a glance on our application domain, AMOSQL queries can for example be stated for retrieving:

- single, multiple, and composite values such as min, max, and intervals of stresses, displacements, etc.
- composition of data can be accomplished by associations, aggregations, constraints, and by extended computations. For example, in a query retrieval it is easy to associate the name of a point or node, its position, and maybe some additional quantity like the stress state. Various weighted quantities, such as von Mises stresses, may not have to be stored in the database. They could instead be computed using system- and user-defined functions accessible in queries. Further examples of these possibilities will be shown in later examples.
- physical (such as mechanical, spatial, and temporal) and abstract data (such as names, versions, and administrative data) can be freely combined within queries.
- user-defined functions can be composed by a combination of system-supported base functions or external functions implemented in some programming language like Java, C, or LISP. Independently on how they were defined they will be transparently available within AMOSQL as any other base function.

Advanced DBMSs like AMOS II can even allow for extensions of the DBMS with new data representations and the ability to extend the query processor with indexes and cost measures to be able to optimize queries that relies on these data representations. Important in this context is mechanisms to handle tailored data representations for numerical data such as arrays and matrices.

3.3 Database Access Using an Array Representation

A first integration of TRINITAS with the AMOS II DBMS replaced the basic FORTRAN-based array representation with a corresponding representation in the database that takes advantage of the AMOS II object-oriented data model. For instance, this adds true object identifiers and AMOS II functions for defining attributes and access methods. This replacement also meant that all data were immediately made accessible at the array level from the query language. Without any further conceptual modelling, it was possible to state queries over all data, including calculated analysis results.

Since all data are available at the query language level, the user has numerous capabilities to compose queries for retrieving, combining, filtering and transforming data. The following query selects a single value from an array[1] of von Mises stresses, where the index corresponds to a certain node:

```
ref(array_named(“VON1  ”),0);
844.779
```

The same query can also be stated more “SQL-ish”, using a syntax similar to standard SQL, as follows:

```
select vonmises
  from array v, real vonmises
    where name(v) = “VON1   ” and
           vonmises = ref(v,0);
```

If one would like to extract the maximum von Mises stress this query can be extended to traverse the complete array and then apply the `max` aggregation operator:

```
select max((select vonmises
  from integer i, integer j, real vonmises
    where
      i = iota(0,j) and
      j = size(array_named(“VON1 ”)) - 1 and
      vonmises = ref(array_named(“VON1 ”),i)));
2269.62
```

Further, it is not necessary to store every type of result in the database. Instead, derived functions can be defined that calculate these measures. The ability to do calculations within queries is exemplified in the following ad-hoc query where the previous query is reformulated in terms of the basic stress components:

```
select max((select vonmises
  from integer i, integer j,
           real sigmaxx, real sigmayy,
           real sigmaxy, real vonmises
  where
    j = size(array_named(“VON1 ”)) - 1 and
    i = iota(0,j) and
    sigmaxx = ref(array_named(“SXX1 ”),i) and
    sigmayy = ref(array_named(“SYY1 ”),i) and
    sigmaxy = ref(array_named(“SXY1 ”),i) and
    vonmises = sqrt(abs(
                 sigmaxx*sigmaxx+
                 sigmayy*sigmayy-
                 sigmaxx*sigmayy+
                 3*sigmaxy*sigmaxy))));
2269.62
```

Associated information can be retrieved by selecting several results in the same query, as in the subsequent query where the stress value is combined with a number corresponding to the node number:

```
select max((select vonmises,i+1
  from integer i, integer j, real vonmises
    where
      i = iota(0,j) and
      j = size(array_named(“VON1 ”)) - 1 and
      vonmises = ref(array_named(“VON1  ”),i)));
<2269.62,83>
```

The query results can further be filtered by adding addi-

1. The `array_named` function maps a name of an array to an array object.

tional constraints in the query. Here, the von Mises stresses that exceed 2000.0 (MPa) are extracted along with the node numbers.

```
select vonmises,i+1
  from integer i, integer j, real vonmises
    where
      i = iota(0,j) and
      j = size(array_named(“VON1  ”)) - 1 and
      vonmises = ref(array_named(“VON1 ”),i) and
      vonmises > 2000.0;
<2269.62,83>
<2128.4,90>
```

The previous examples showed how the current array representation was directly accessible from the query language. In comparison to the original TRINITAS program, the expressibility for accessing and composing data have increased dramatically in FEAMOS, even if the queries became a bit clumsy on occasion. In addition, it is not so critical if some relevant evaluation functionality has been overlooked in the implementation. It will be relatively simple to add this through the query language. In TRINITAS, this must be done by implementing some new FORTRAN routines and recompile at least part of the system. An alternative would be to use a macro language to specify user-defined functionality, if such a language is available to the application and can be used for that purpose.

3.4 Database Access Using a Domain Model

By introducing more structure within analysis result information within the database and through the AMOSQL object-oriented data model, information can be retrieved very conveniently. We call this usage of a *domain model* that is a conceptualization and database representation of concepts and operations of the application domain. As an example the information about the stress fields can be associated to the nodes and we can take a look at how to retrieve von Mises stresses using a threshold value. This simple example include a selection of nodes constrained by the condition that the von Mises stress should not exceed 1500.0 MPa. In the AMOSQL language this could be expressed as:

```
select n
  from node n
    where vonmises(n) > 1500.0;
```

Hence, the query expression retrieves (`select`) the set (sometimes a multiset) of `n` objects from the set of objects that is of type `node` and that fulfils the selection condition (the `where` clause). The selection condition, i.e. `“vonmises(n)> 1500.0”`, constrains the selection to those nodes that have a von Mises stress that do not exceed 1500.0 [MPa]. Additional constraints can easily be introduced in the where clause of the query.

As was pointed out earlier, the examples in the previous section implied a rather clumsy and unattractive use of the query language. Further, to get rid of this disadvantage, one could introduce much more structure within the result information through object-oriented modelling. For instance, by associating the stress components to the nodes, the former queries can be expressed and interpreted much more conveniently. This can be accomplished as:

```
create function stresses(node n) -> farray(3) as
        stored;
```

As we only consider stress components in the xy-plane, the `stresses` function stores an array of three reals that can represent the stress components σ_{xx}, σ_{xy}, and σ_{yy}. These can be explicitly referenced and defined by derived functions.

```
create function sigmaxx(node n) -> real as
        select ref(stresses(n),0);

create function sigmaxy(node n) -> real as
        select ref(stresses(n),1);

create function sigmayy(node n) -> real as
        select ref(stresses(n),3);
```

It is further possible to add functions that improves the intuition of the domain without wasting any storage space. For instance, we can add an alias derived function for the σ_{yx} component:

```
create function sigmayx(node n) -> real as
        select sigmaxy(n);
```

We can even add some alias functions if we rather would like to use “tao” (τ) to refer to the skew components.

```
create function taoxy(node n) -> real as
        select sigmaxy(n);

create function taoyx(node n) -> real as
        select sigmayx(n);
```

Since these functions are compiled and optimized when they are defined, there is no performance loss for using these alias functions. The function for computing the von Mises stress can now be expressed as follows:

```
create function vonmises(node n) -> real vm as
        select sqrt(abs(
                sigmaxx(n)* sigmaxx(n)+
                sigmayy(n)* sigmayy(n)-
                sigmaxx(n)* sigmayy(n)+
                3*taoxy(n)* taoyx(n)));
```

By structuring the stresses along these lines, the previous query for retrieving the maximum von Mises stress can now be expressed as:

```
select max((select vonmises(n)
          from node n));
2269.62
```

Likewise, the node[1] can be associated and retrieved together with the von Mises stress:

```
select max((select vonmises(n),n
          from node n));
<2269.62,OID[0x0:1543]>
```

Additional constraints can easily be introduced in a similar manner as in the former example, but here there is a more direct association to the stress function.

1. Note that the node is here an object and not a number as in the previous example.

```
select vonmises(n)
  from node n
    where vonmises(n) > 2000.0;
2269.62
2128.4
```

It is also easy to express queries that include more complex constraints. This will be illustrated by means of another example shown in Figure 2. The following query retrieves nodes and their corresponding von Mises stresses within a certain area:

```
select n, vonmises(n)
  from node n
    where x_coordinate(n) >= 50.0 and
           x_coordinate(n) <= 70.0 and
           y_coordinate(n) >= 40.0 and
           y_coordinate(n) <= 60.0;
<OID[0x0:1708],630.798>
<OID[0x0:1712],1982.9>
<OID[0x0:1713],958.276>
<OID[0x0:1714],876.571>
<OID[0x0:1715],1728.37>
<OID[0x0:1716],1310.12>
```

Here, it is presupposed that coordinate functions are defined for nodes. As pointed out earlier, query expressions can sometimes be easier to interpret visually by alias functions. In a mathematical expression it can for example be clearer to use a more mathematically-oriented notation such as:

```
select n, x(n), y(n), vonmises(n)
  from node n
    where x(n) >= 50.0 and
           x(n) <= 70.0 and
           y(n) >= 40.0 and
           y(n) <= 60.0;
<OID[0x0:1708],70.,40.,630.798>
<OID[0x0:1712],53.6603,43.6603,1982.9>
<OID[0x0:1713],60.406,47.935,958.276>
<OID[0x0:1714],70.,55.,876.571>
<OID[0x0:1715],50.,50.,1728.37>
<OID[0x0:1716],60.,60.,1310.12>
```

Switching to infix notation in AMOSQL can further increase the readability:

```
select vonmises(n)
  from node n
    where 50.0 <= x(n) <= 70.0 and
           40.0 <= y(n) <= 60.0;
```

While still looking at the example in Figure 2, the subsequent query shows how to select nodes and stress components along a horizontal line with a specific y-coordinate such as:

```
select n,x(n),sigmaxx(n),sigmayy(n),sigmaxy(n)
  from node n
    where y(n) = 0.0;
<OID[0x0:1678],0.,-766.462,1497.83,5.11513>
<OID[0x0:1681],15.,-841.216,484.575,187.309>
<OID[0x0:1686],30.,-953.525,-1267.17,-330.958>
```

Yet another query, also corresponding to Figure 2, shows how to select nodes and the von Mises stress along the line with the equation $y(x) = x$:

```
select n, x(n), y(n), vonmises(n)
  from node n
    where y(n) = x(n);
<OID[0x0:1678],0.,0.,1994.76>
<OID[0x0:1680],10.,10.,1152.1>
<OID[0x0:1684],20.,20.,936.039>
<OID[0x0:1690],30.,30.,2342.41>
<OID[0x0:1715],50.,50.,1728.37>
<OID[0x0:1716],60.,60.,1310.12>
<OID[0x0:1717],70.,70.,1171.47>
```

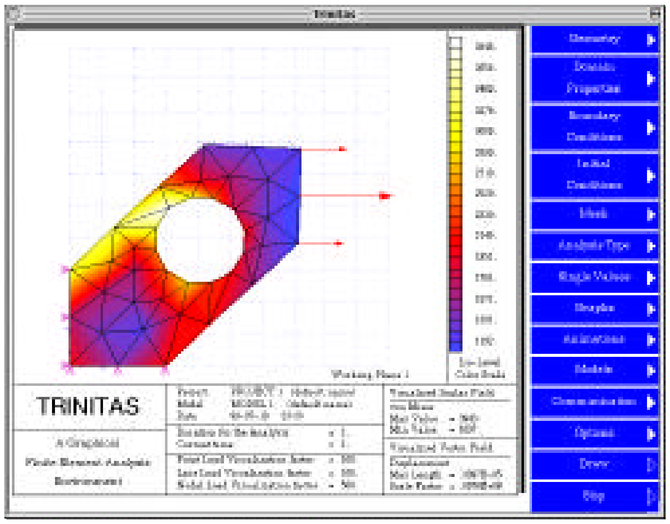


Figure 2. *An analysis model used to show retrieval of analysis data constrained by various geometric conditions.*

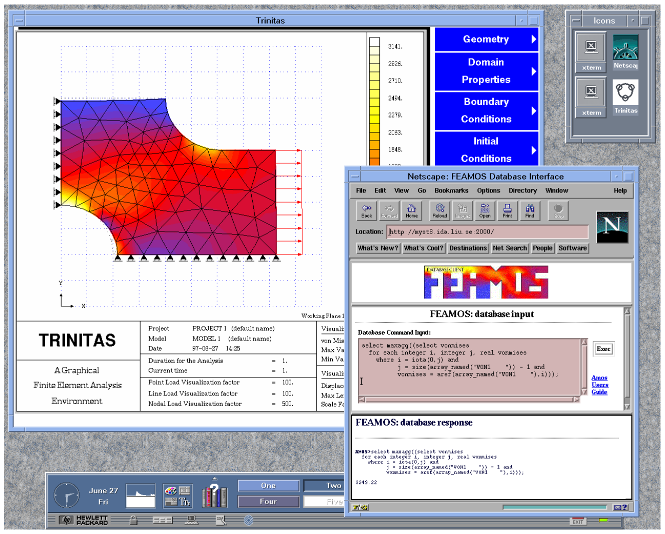
Functions can further be overloaded on several other types (or combinations of types) to add capabilities to retrieve a wealth of additional analysis information. For example, by overloading stress functions on other types, in addition to nodes, it is possible to select stresses on boundaries, curves, surfaces, or other geometrical concepts. A general capability to retrieve values from continuous field quantities at arbitrarily spatial positions require that interpolation of discrete data is handled appropriately. An interesting area related to this matter is the potential need for indexing of interpolated data as treated in Lin [18, 19].

3.5 User Interaction Through Queries

A user of the FEAMOS system can currently use a web-browser as a database client to directly interact with the FEAMOS database through the query language as shown in Figure 3. This web-based database client uses a general web-server-interface implemented in AMOS II to compose queries and retrieve query results textually.

Furthermore, in a recent contribution to the AMOS II project a Java programming interface and a database browser have been developed to support development of AMOS II-based database systems.

Current studies investigates possible solutions on how to implement mechanisms to support graphical query response, i.e. capabilities to automatically display query results graphically whenever it is relevant. This means that the user should be able to evaluate a graphical representation of the result of a query instead of having to browse through a large set of textual result tuples. This include an ability to present spatial information in an appropriate form which is of great importance for various scientific and engineering applications.



Project PROJECT 1 (default name)
Model MODEL 1 (default name)
Date 97-06-27 14:25
Duration for the Analysis = 1.
Current time = 1.
Point Load Visualization factor = 100.
Line Load Visualization factor = 100.
Nodal Load Visualization factor = 500.

http://myst8.ida.liu.se:2000/

FEAMOS: database input

Database Command Input:

```
select maxagg((select vonmises
  for each integer i, integer j, real vonmises
    where i = iota(0,j) and
          j = size(array_named("VON1    ")) - 1 and
          vonmises = aref(array_named("VON1    "),i)));
```

FEAMOS: database response

```
AMOS>select maxagg((select vonmises
  for each integer i, integer j, real vonmises
    where i = iota(0,j) and
          j = size(array_named("VON1    ")) - 1 and
          vonmises = aref(array_named("VON1    "),i)));

3249.22
```

Figure 3. *An illustration of the use of a www-based database client in combination with FEAMOS for providing ad-hoc query capabilities.*

4 Summary

It has been shown in this paper how it is possible to use an extensible and object-oriented query language to access and manipulate FEA results. This makes it possible to leverage on general computing mechanisms provided by modern database technology that otherwise would have had to be specifically implemented for the application. The advantages of this approach are manifold including an enhanced flexibility of the data resources and even a potential capability of improving the processing efficiency [4].

More specifically, the *accessibility* has increased dramatically in FEAMOS where all data are available at the query language level. It is neither as critical if some relevant functionality for evaluating the analysis has been overlooked in the implementation, since it is very simple to add this through the query language. In conventional applications like TRINITAS, this must normally be done by implementing some new programming procedures and recompile the system.

Furthermore, by introducing more conceptualization through the domain models, the *expressibility* for composing data also increased considerably. The query language facilitates *retrieving*, *combining*, *constraining*, and *transforming* of data through its general and declarative capabilities for posing queries and the abilities to incorporate extensions into the language.

The author also argues that a more high-level and declarative expression of domain models provide a more *intuitive* and *simpler* representation that is easier to understand than conventional and more procedural-oriented representations. Simpler models will eventually lead to better quality and higher security in the design and analysis process. Simplicity will also result in more efficient development, use, and maintenance (reuse) of these domain models. By relying on general mechanisms for optimizing data access one will also facilitate an efficient use of computational resources - especially when EISs become more and more complex.

In a wider perspective, the introduction of database technology into computational scientific and engineering applications will provide a sound base for supporting data *mediation* in the EIS environment. This include capabilities to *transform*, *integrate*, *separate*, *locate*, and *monitor* data to be able to keep data consistency while *sharing*, *exchanging*, and *combining* data in a heterogeneous and distributed EIS environment.

Within this context there are also two database mechanisms that ought to be studied in further research. As indicated in other areas of FEA, indexes can be of great importance for supporting efficient processing. FEA results, such as stress and temperature fields, are examples of spatial quantities where *spatial indexing techniques* [18, 19] could be applied. Furthermore, in studying the issue of whether a stored or a derived representation of analysis results is preferable, it should be of interest to evaluate how *function materialisation techniques* [17] could support these decisions in an automatic manner.

Even if the user can get great support from graphics and a query language, as described here, the result interpretation is mainly a manual activity. The efficiency and quality could probably be increased for this activity by taking advantage of the rule system in AMOS II. By defining *active rules* [20, 21] that interpret the results, parts of the evaluation activity could be automated. The applicability of these ideas also requires additional study.

5 Acknowledgements

This research has been sponsored by The Swedish Foundation for International Cooperation in Research and Higher Education and by The Swedish National Board for Industrial and Technical Development.

The author is currently on leave from The Engineering Databases and Systems Laboratory at The Department of Computer and Information Science, Linköping University, Linköping, Sweden.

Bibliography

[1] Cattell, R. G. G., “Object Data Management: Object-Oriented and Extended Relational Database Systems”, Addison-Wesley, 1992.

[2] Orsborn, K., “Applying Next Generation Object-Oriented DBMS to Finite Element Analysis”, 1st Int. Conf. on Applications of Databases, ADB94, Lecture Notes in Computer Science, 819, Springer Verlag, 1994, p. 215-233.

[3] Orsborn, K. and Risch, T., “Next Generation of O-O Database Techniques in Finite Element Analysis”, 3rd Int. Conf. on Computational Structures Technology (CST96), Budapest, Hungary, August, 1996.

[4] Orsborn, K., “On Extensible and Object-Relational Database Technology for Finite Element Analysis Applications”, PhD Thesis (Linköping studies in science and technology. Dissertations 452), ISBN 91-7871-827-9, Linköping University, Linköping, Sweden, October 1996.

[5] Flodin, S., Orsborn, K., and T. Risch, “Using Queries with Multi-Directional Functions for Numerical Database Applications”, Second East-European Symposium on Advances in Databases and Information Systems (ADBIS'98), Poznan, Poland, September 1998.

[6] Tisell, C., and Orsborn, K. “A System for Multibody Analysis Based on Object-Relational Database Technology”, The 1st Intl. Conf. on Engineering Computational Technology (ECT98), Edinburgh, Scotland, August 1998.

[7] Flodin, S., Josifovski, V., Risch, T., Sköld, M. and Werner, M., “AMOS II User's Guide”, available at http://www.ida.liu.se/˜edslab.

[8] Fahl, G., Risch, T. and Sköld, M., “AMOS, an Architecture for Active Mediators”, Int. Workshop on Next Generation Information Techn. and Systems, Haifa, Israel, June 1993.

[9] Torstenfelt, B., Allestam, H., and Klarbring, A., “Shape Optimization Implemented in an Object-Oriented Finite Element Program Environment”, 6th Nordic Seminar on Computational Mechanics, Linköping, 1993.

[10] Torstenfelt, B., “An Integrated Graphical System for Finite Element Analysis”, User’s Manual Version 2.0, LiTH-IKP-R-737, Linköping University, Linköping, January 1993.

[11] Eisenberg, A. and Melton, J., “SQL:1999, Formerly Known as SQL3”, SIGMOD Record, v. 28, n. 4, March 1999, p. 131-138.

[12] Cattell, R. G. G., et al., “The Object Database Standard: ODMG 2.0”, Morgan Kaufmann Publishers, Inc., 1997.

[13] Stonebraker, M. and Moore, D., “Object-Relational DBMSs: The Next Great Wave”, Morgan Kaufmann Publishers, Inc., 1996.

[14] Wiederhold, G. and Genesereth, M., “The Conceptual Basis for Mediation Services”, IEEE Expert, Intelligent Systems and their Applications, v. 12, n. 5, October 1997, p. 38- 47.

[15] Josifovski, V., “Design, Implementation and Evaluation of a Distributed Mediator System for Data Integration”, PhD Thesis (Linköping studies in science and technology. Dissertations 582) ISBN 91-7219-482-0, Linköping University, Linköping, Sweden, June 1999.

[16] Josifovski, V., and Risch, T., “Integrating Heterogeneous Overlapping Databases Through Object-Oriented Transformations”, to be presented at 25th Intl. Conf. On Very Large Databases, Edinburgh, Scotland, September 1999.

[17] Kemper, A., Kilger, C., and Moerkotte, G., “Function Materialization in Object Bases: Design, Realization, and Evaluation”, IEEE Transactions on Knowledge and Data Engineering, v. 6 n. 4, August 1994, p. 587-608.

[18] Lin, L. and Risch, T., “Querying Continuous Time Sequences”, 24th International Conference on Very Large Data Bases (VLDB'98), New York City, USA, August 1998.

[19] Lin, L., “Management of 1-D Sequence Data - from Discrete to Continuous”, PhD Thesis (Linköping studies in science and technology. Dissertations 561), ISBN 91-7219-402-2, Linköping University, Linköping, Sweden, March 1999.

[20] Sköld, M. and Risch, T., “Using Partial Differencing for Efficient Monitoring of Deferred Complex Rule Conditions”, In The 12th International Conference on Data Engineering (ICDE’96), (IEEE), New Orleans, Louisiana, February 1996.

[21] Sköld, M., “Active Database Management Systems for Monitoring and Control”, PhD Thesis (Linköping studies in science and technology. Dissertations 494), ISBN 91-7219-002-7, Linköping University, Linköping, Sweden, September 1997.