OODBMS: A Case Study. IEEE Computer Soc. Int. Conf. 35 San Francisco 1990 Digest of papers/ Compcon spring 90, February 26 - March 2, 1990, 528-537.

18. Cook, R. D.: Concepts and Applications of Finite Element Analysis. 3rd, John Wiley & Sons, Inc., 1989.
19. Atkinson, M., Bancilhon, F., DeWitt, D., Dittrich, K., Maier, D., Zdonik, S.: The Object-Oriented Database System Manifesto. in Kim, W., Nicolas, J-.M., Nishio, S., eds., Proceedings of the First International Conference on Deductive and Object-Oriented Databases (DOOD), Elsevier Science Publishers, Amsterdam, 1989, 40-57.
20. The Committee for Advanced DBMS Function: Third-Generation Database System Manifesto. SIGMOD Record, **19**(3), September 1990, 31-44.
21. Litwin, W., Risch, T.: Main Memory Oriented Optimization of OO Queries using Typed Datalog with Foreign Predicates. IEEE Transactions on Knowledge and Data Engineering, **4**(6), December 1992, 517-528.
22. Wolniewicz, R., Graefe, G.: Algebraic Optimization of Computations over Scientific Databases. Proceedings of the 19th VLDB Conference, Dublin, Ireland, August 24-27, 1993, 13-24.
23. Takizawa, M.: Distributed Database System JDDBS. JARECT Computer Science & Technologies. **7**, OHMSHA & North Holland (publ.), 1983, 262-283.
24. Catell, R. G. G.: Object Data Management: Object-Oriented and Extended Relational Database Systems. Addison-Wesley Publishing Company, Inc., 1991 (reprinted with corrections 1992).
25. Abiteboul, S., Bonner, A.: Objects and Views. Proceedings of the ACM SIGMOD Conference, 1991, 238-247.
26. Bancilhon, F., Delobel, C., Kanellakis, P. (eds.): Building an Object-Oriented Database System: The Story of O2. Morgan Kaufmann Publishers, Inc., 1992.
27. Risch, T., Sköld, M.: Active Rules Based on Object-Oriented Queries. LiTH-IDA-R-92-35, Linköping University, 1992. Also in a special issue on Active Databases of IEEE Data Engineering, **15**(1-4), December, 1992.
28. Fishman, D. H., Annevelink, J., Chow, E. Connors, T., Davis, J. W., Hasan, W. Hoch, C. G., Kent, W., Leichner, S., Lyngbaek, P., Mahbod, B., Neimat, M.A., Risch, T., Shan, M. C., Wilkinson, W. K.: Overview of the Iris DBMS. in Kim, W., Lochovsky, F. H. (eds.): Object-Oriented Concepts, Databases, and Applications, ACM Press, Addison-Wesley, 1989, 219-250.
29. Shipman, D. W.: The Functional Data Model and the Data Language DAPLEX. ACM TODS, **6**(1), March 1981, 140-173.
30. Karlsson, J., Larsson, S., Risch, T., Sköld, M., Werner, M.: AMOS Users's Guide., CAELAB Memo 94-01, Linköping University, March 1994.

References

1. French, J. C., Jones, A. K., Pfaltz, J. L.: Summary of the Final Report of the NSF Workshop on Scientific Database Management. SIGMOD Record, **19**(4), December 1990, 32-40.
2. IEEE Computer Society, The Bulletin of the Technical Committee on Data Engineering (TCDE), Special Issue on Scientific Databases, **93**(2), 1993.
3. DBMS: A New Direction in DBMS. Interview with Michael R. Stonebraker in DBMS, February 1994, 50-60.
4. Fahl, G., Risch, T., Sköld, M.: AMOS - An Architecture for Active Mediators. The International Workshop on Next Generation Information Technologies and Systems (NGITS' 93), Haifa, Israel, June 28-30, 1993, 47-53.
5. Torstenfelt, B.: An Integrated Graphical System for Finite Element Analysis, User's manual. Version 2.0, LiTH-IKP-R-737, Linköping University, January 1993.
6. Lyngbaek, P.: OSQL: A Language for Object Databases. HPL-DTD-91-4, Hewlett-Packard Company, January 1991.
7. Beech, D.: Collections of Objects in SQL3. Proceedings of the 19th VLDB Conference, Dublin, Ireland, August 24-27, 1993, 244-255.
8. Becker, E. B., Carey, G. F., Oden, J. T.: Finite Elements: An Introduction. Prentice-Hall, Inc., Vol. 1, Texas Finite Element Series, 1981.
9. Baugh, J. W., Rehak, D. R.: Object-Oriented Design of Finite Element Programs. Computer Utilization in Structural Engineering Proceedings of the Sessions at Structures Congress '89, San Francisco, CA, USA, May 1-5, 1989, 91-100.
10. Fenves, G. L.: Object-Oriented Programming for Engineering Software Development. Engineering with Computers **6**, 1990, 1-15.
11. Forde, B. W. R., Foschi, R., Stiemer, S. F.: Object-Oriented Finite Element Analysis. Computers & Structures **34**(3), 1990, 355-374.
12. Filho, J. S. R. A., Devloo, P. R. B.: Object-Oriented Programming in Scientific Computations: the Beginning of a New Era. Engineering Computations **8**, 1991, 81-87.
13. Dubois-Pelerin, Y., Zimmermann, T., Bomme, P.: Object-Oriented Finite Element Programming: II. A Prototype Program in Smalltalk. Computer Methods in Applied and Engineering **98**, 1992, 361-397.
14. Williams, J. R., Lim, D., Gupta, A.: Software Design of Object Oriented Discrete Element Systems. Proceedings of the Third International Conference on Computational Plasticity, Barcelona, Spain, April 6-10, 1992, 1937-1947.
15. Tworzydlo, W. W., Oden, J. T.: Towards an Automated Environment in Computational Mechanics. Computer Methods in Applied and Engineering **104**, 1993, 87-143.
16. Ahmed, S., Wong, A., Sriram, D., Logcher, R.: Object-Oriented Database Management Systems for Engineering: A Comparison. Journal of Object-Oriented Programming, **5**(3), June 1992, 27-44
17. Ketabchi, M. A., Mathur, S., Risch, T., Chen, J.: Comparative Analysis of RDBMS and

- Extensibility of a query language, such as AMOSQL, will also make it possible to make optimizations over the domain model and thereby domain-related operations.

Future work will investigate these issues in greater detail. This also involves an inclusion of matrices and matrix procedures for modelling numerical algorithms.

Thus, using next generation OO DBMS and extensible OO query languages can renew development, maintenance, and usage of FEA software. The benefits of using domain models include easier access through a query language, better data description (as schemas), and other benefits currently provided only by advanced DBMS, such as transaction capabilities and ad hoc query processing. We believe that tools like object-relational database management systems and extensible query languages can provide a solid base for future scientific and engineering data management.

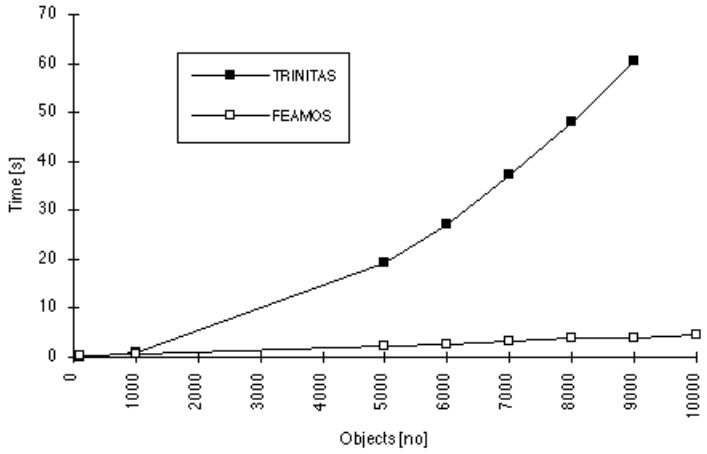

Fig. 9. Performance measures of TRINITAS and FEAMOS. This diagram shows the real access time for accessing the position of point objects

3 Summary

Results from the initial implementation of the FEA application treated in this paper show that the use of an OR DBMS including an extensible OO query language is a promising direction/competitive alternative in designing future FEA software. Both requirements on domain modelling and execution efficiency can be met by this approach.

High-level declarative modelling is supported by the extensible OO query language which makes development, maintenance, and use of the applications more effective. Furthermore, initial performance measures show that the overhead of using an embedded database is quite low, in particular if the database can be run in the same address space as the application. More specifically,

- It is not necessary to re-implement low-level dedicated data structures, such as indices, for each new system. Such re-implementation not only duplicates implementation efforts but, as our example shows, may prove less efficient than the highly optimized data management provided by an embedded DBMS.
- By providing access to other databases, e.g. relational DBMS [4], from the domain model it is possible to build models and ad hoc queries that combine data from other databases, e.g. from other components of a CAE system.
- Domain models can provide a generic model, here a FEA model, that can be used by many applications, providing design re-use and simplifying the combination of data from several systems.

TRINITAS system has acceptable response times and processing efficiency when it is used. The performance test measured the time for creation of objects including an initialization (update) of a stored function. In this case this corresponds to point objects and a position attribute with three coordinates. We also measured the access time for random access of the same number of objects including the function access. These results are presented in Fig. 9. The access phase is more critical than the creation phase since it has a higher frequency in a real application situation.

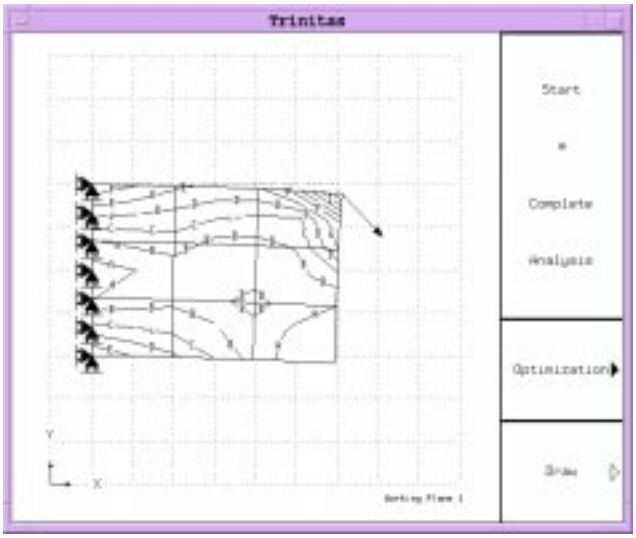

Fig. 8. A contour plot for the example showing iso-stress curves resulting from a linear static stress analysis

The improved performance of FEAMOS compared to TRINITAS is explained by the lack of dedicated storage structures in TRINITAS. TRINITAS performs linear search over the point object set, whereas FEAMOS takes advantage of built-in storage structures of AMOS, such as hash tables, for efficient access. These kinds of storage structures can, of course, also be implemented in TRINITAS, which should result in similar inclination of the TRINITAS performance curves. At small sets of objects (about <500), TRINITAS has better performance than FEAMOS which is due to the interface overhead in accessing the database. However, the processing performance is no real bottle-neck at these object volumes.

The quite encouraging performance measures of FEAMOS are also due to the fact that the database is embedded (shares the same address space) in the application and that the database is in main-memory. Disk access and process or network communication would severely slow down the system.

point to a specific position, calculating distances between positions and points, etc. For this purpose, it would also be interesting to investigate how spatial indexing techniques could be used to support efficient processing of these types of queries. However, these operations can also be transferred into the database. These operators are expressed in AMOSQL as:

```
create function distance(real x1, real y1, real z1,
                         point p2) -> real d as
          select d for each real x2, real y2, real z2,
                            real t1, real t2, real t3
             where
                position(p2) = <x2, y2, z2> and
                t1 = (x2 - x1) and
                t2 = (y2 - y1) and
                t3 = (z2 - z1) and
                d = sqrt((t1*t1) + (t2*t2) + (t3*t3));
create function find_nearest_point(
                      real x, real y, real z) -> point p1 as
          select p1
             where distance(x, y, z, p1) =
                       minagg((select distance(x, y, z, p2)
                          for each point p2));
```

And an application of these functions looks like:

```
distance(100.0, 100.0, 0.0, :point_3);
<22.3607>
name(find_nearest_point(100.0, 100.0, 0.0));
<“p3”>
```

As the implementation continues, it will be possible to transfer more and more functionality to the database and large parts of the processing can be kept within the database system. The database model will be further developed to eventually include a complete FEA. This includes modelling of concepts and functionality related to the finite element mesh, domain properties, boundary conditions, the actual analysis, and result interpretation. Some simple examples of the concepts and relations that are apparent for the mesh was illustrated in Sect. 2.3. The analysis step involves the solution of a system of equations that requires the representation of matrices and matrix operators in the database. The processing of numerical calculations can be made by extending the query language with numerical operators as foreign functions.

The current implementation, FEAMOS, has also been evaluated with respect to its execution performance. FEAMOS has been compared with TRINITAS, the original FEA program. Since execution performance is quite critical for these kinds of systems it is important that new software design principles are able to scale up with the application. To traditionalists it might be surprising that the evaluation result showed that the FEAMOS system scaled up better than TRINITAS. Especially, since the original

The `edges` function is modelled to store object identifiers internally for generality and efficiency reasons. However, the `name` function can also be used on each element of the vector for name reference. The preceding example would then look like:

```
select name(elements(edges(:surface_1)));
<“line_1”> <“line_2”> <“line_3”> <“line_4”>
```

It is also possible to invert functions, i.e. apply the function in the opposite direction as in the following query that tells which surfaces a specific edge belongs to:

```
select name(s) for each surface s
               where elements(edges(s)) = :line_2;
<“s1”>
```

In TRINITAS, the modelling of the example results in a geometry model as in Fig. 7. The geometry model is thereafter used as the basis for the specification and generation of a finite element mesh. If we specify a mesh with three bilinear elements per edge of the rectangular the resulting mesh is shown in Fig. 3. Figure 8 presents a view of the results from a linear static stress analysis of the model where a contour plot of iso-stress curves is included.

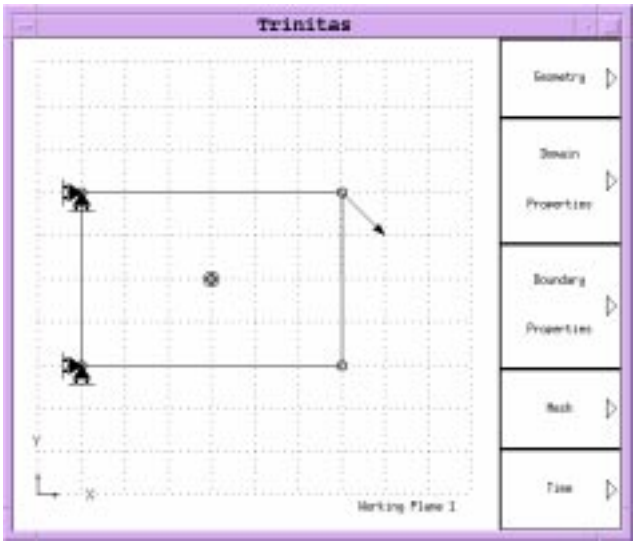

Fig. 7. The geometry model, of the example, with boundary conditions modelled in TRINITAS

The functionality of geometry-related concepts that are related to the user interface management is currently not modelled in the database. This includes finding the nearest

The modelling of a geometry can be illustrated by the example shown earlier in Fig. 2. The resulting model is built by basic geometric elements, i.e. 4 points, 4 straight lines, 1 surface, and 1 volume elements, shown in Fig. 6. This can be expressed in AMOSQL as:

```
create point(name, position)
         :point_1("p1", <0.0, 0.0, 0.0>),
         :point_2("p2", <0.0, 120.0, 0.0>),
         :point_3("p3", <90.0, 120.0, 0.0>),
         :point_4("p4", <90.0, 0.0, 0.0>);

create straight_line(name, vertices, division, density)
         :line_1("l1", {:point_1, :point_2}, 3.0, 0.0),
         :line_2("l2", {:point_2, :point_3}, 3.0, 0.0),
         :line_3("l3", {:point_3, :point_4}, 3.0, 0.0),
         :line_4("l4", {:point_4, :point_1}, 3.0, 0.0);

create surface(name, edges) :surface_1("s1",
         {:line_1, :line_2, :line_3, :line_4});

create volume(name, faces) :volume_1("v1", {:surface_1});
```

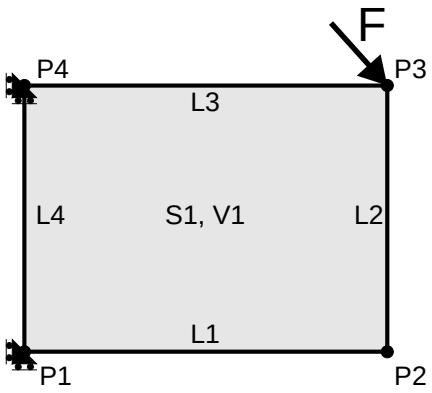

Fig. 6. Basic geometrical model of the structure in Fig. 2 including boundary conditions

This modelling technique then makes it possible to put queries to the model about its structure and content, i.e. basic geometrical and topological information in this case.

For example the edges of `:surface_1` can be extracted by:

```
select edges(:surface_1);
<{OID[0x0:294], OID[0x0:295], OID[0x0:296], OID[0x0:297]}>
```

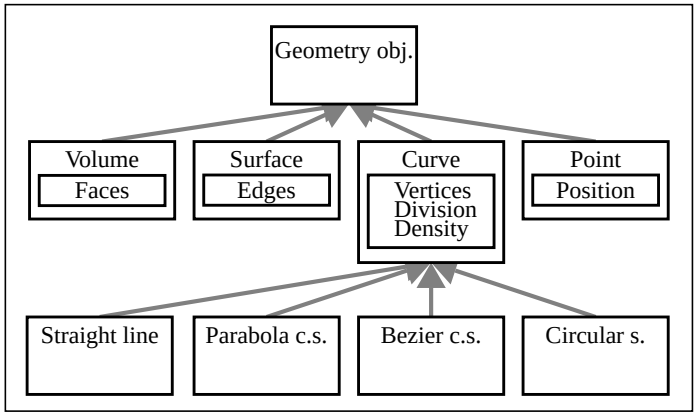

Fig. 4. Partial type structure for the geometry-related subset of the concept domain. Arrows denote is-a (subtype to supertype) relations

When a geometry is modelled in TRINITAS, an object structure is then generated in an AMOS database by means of interface functions defined in AMOSQL providing encapsulation and data independence. Likewise, a manipulation of an object, as moving a point on the screen, implies a direct update of the database object. There are, for instance, functions and procedures for *constructing* and *destructing* objects as well as for *accessing* and *updating* functions.

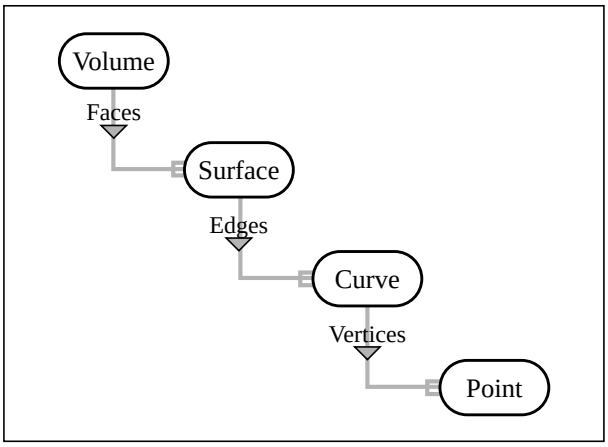

Fig. 5. Topological relations between geometry concepts

Deletion of types, functions, and objects is made through a `delete` statement as:

```
delete type element;
delete function nodes;
delete :element_1;
```

In addition to database population by object creation and attribute assignments it is possible to use *function update statements* `set`, `add`, and `remove`, and *type update statements* `add` and `remove`. Examples of update statements for functions are:

```
set nodes(:element_1) = <:node_1, :node_2, :node_4>;
add nodes(:element_1) = :node_5;
remove nodes(:element_1) = :node_2;
```

A more complete presentation of data management capabilities in AMOS and AMOSQL is presented in [30].

2.4 Domain Modeling of Finite Element Analysis in FEAMOS

An initial integration of TRINITAS and AMOS has been implemented. Data structures and corresponding procedures implemented in TRINITAS (written in FORTRAN) are incrementally replaced by schemas and operators in AMOS. TRINITAS has originally been designed in a highly structured, "object-based", manner with specific sets of procedures for each concept class, such as point, line, etc. This makes it easier to replace subsets of the FEA program part by part incrementally, and a demonstrable system exists at every stage. The programs are linked together and communication is done by transferring parameters between FORTRAN and C procedures using the AMOS C-interface. Concurrently with the normal TRINITAS interface it is possible to query the contents in the database through a database monitor that currently is a standard textual AMOS window.

Currently, the representation of the FEA domain in AMOS mainly covers geometry-related concepts. The classification of these are presented in Fig. 4. An abstract class `geometry_object` holds the basic geometry classes that include `volume, surface, curve,` and `point`. Corresponding subclasses in TRINITAS exists for the `curve` class and include `straight_line, parabola_cubic_section, circular_segment,` and `bezier_cubic_segment`. All these classes are modelled as types and subtypes in AMOS.

Topology relations `faces, edges,` and `vertices` are modelled as stored functions between basic geometry classes as illustrated in Fig. 5. For example, the function `vertices` defined by

```
create function vertices(curve c) -> vector as stored;
```

provides a relation from a curve instance to its points. In addition, the `curve` type currently has a `division` and a `density` function implemented. The `division` function represents the number of subdivisions a specific curve is divided into and the `density` function represents a node density along a curve. The `point` class has a function `position` that stores the x, y, and z spatial coordinates of a point instance.

```
name(topology(:element_1));
<“e2”> <“e4”> <“e5”>

name(topology(:element_5));
<“e1”> <“e2”> <“e4”> <“e6”> <“e8”> <“e9”> <“e7”> <“e3”>
```

Querying a database for objects having specified properties is made using a `select` statement. For instance the `nodes` of `:element_1` in the example earlier can be retrieved by the following query:

```
select name(nodes(:element_1));
<“n1”> <“n6”> <“n5”> <“n2”>
```

Functions are also invertible (not always) and it is therefore possible to use the `nodes` function in the opposite direction which can be expressed as:

```
select name(e) for each element e where nodes(e) = :node_1;
```

Another example shows how boundary conditions defined on the geometry, as the fixed edge in Fig. 3, can be identified on the mesh by a connectivity function. All elements affected by this boundary condition can then be retrieved as follows:

```
select unique name(e) for each element e, node n, curve c where
            name(c) = “l4” and nodes(e) = n
            nodal_to_curve_connectivity(n) = c;
<“e1”> <“e4”> <“e7”>
```

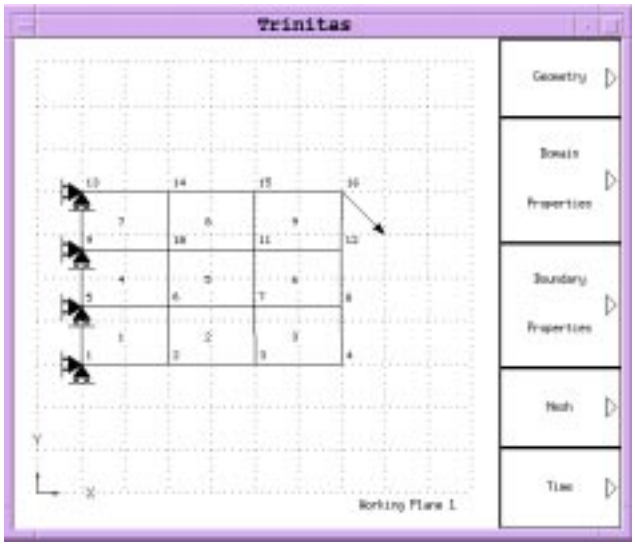

Fig. 3. A simple FE mesh consisting of 9 bilinear elements including node and element numbers. Rigid boundary conditions are introduced for the left edge and the loading condition is modelled by a point load. Note that node and element numbers are included only for facilitating interpretation of the examples and is not required (but optional) by the FEAMOS system

Data schemas can be defined, modified, and deleted by means of AMOSQL statements both statically and dynamically. The definition of types, functions, and objects is made through a `create` statement. For example, types may be defined by a `create type` statement as:

```
create type named_object(name charstring);
create type fea_object subtype of named_object;
create type element subtype of fea_object;
create type node subtype of fea_object;
```

where a stored function, `name`, is defined within the parentheses of the `named_object` type. A new type becomes an immediate subtype of all supertypes provided in the `subtype` clause, or if no supertypes are specified, it becomes an immediate subtype of the system type `UserTypeObject`.

Functions can also be defined separate from the types by a `create function` statement, exemplified by the `nodes` function that relates elements to nodes:

```
create function nodes(element e1) -> node n as stored;
```

A database for the example in Fig. 3 is populated with objects with a `create` *`type`* statement with or without initialization of functions, and where *`type`* stands for the specific type to be instantiated. The database example can be created by the following statements:

```
create node (name)    :node_1 (“n1”),
                      :node_2 (“n2”),
                      ...
                      :node_16 (“n16”);
create element (name, nodes)
   :element_1 (“e1”, <:node_1, :node_2, :node_6, :node_5>),
   :element_2 (“e2”, <:node_2, :node_3, :node_7, :node_6>),
                      ...
   :element_9 (“e9”, <:node_11, :node_12, :node_16, :node_15>);
```

Derived functions are defined in a similar manner as stored functions with a single AMOSQL-query as the function body. An example of a derived function is presented as the `topology` function below:

```
create function topology(element e1) -> element e2 as
      select unique e2
            for each element e2
               where nodes(e1) = nodes(e2) and
                     e1 != e2;
```

The topology can be identified in Fig. 3, and defines how elements are related to each other. When the `topology` function is accessed, it derives the elements `e2` that have some common node with element `e1`, i.e. the elements that are connected to a given element. An example shows the topology for element 1 and 5 respectively.

skin panel or wing in an aircraft design, finite elements, geometrical elements, etc. System-specific objects, e.g. types and functions, are also treated as surrogate objects.

Types are used to structure objects according to their functional characteristics, in other words it is possible to structure objects into types. Types are in themselves related in a type hierarchy of subtypes and supertypes. Subtypes inherit functions from supertypes and can have multiple supertypes. In addition, functions can be overloaded on different subtypes (i.e. having different implementation for different types).

Functions are defined on types, and are used to represent attributes of, relationships among, and operations on objects. Examples of functions for these different categories might be diameter, distance, and move_point. It is possible to define functions as *stored, derived, procedure* or *foreign*. A stored function has its extension explicitly stored in the database, whereas a derived, procedure, or a foreign function has its extension defined in an AMOSQL query, an AMOSQL procedure, or a function in an external language, respectively. Furthermore, functions can be defined as one- or many-valued and are invertible when possible. Stored and some derived functions can be explicitly updated using update semantics but other functions need special treatment for updates.

Data Management in AMOSQL. AMOSQL provides statement constructs for typical database tasks, including data definition, population, updates, querying, flow control, and session and transaction control. Selected parts of these constructs is presented by means of the example in Fig. 2, which will be used in subsequent sections. The figure shows a rectangular plate that is fixed on a wall at its left side and is further exposed to a tension load through a wire connected at the upper right corner. A simple finite element model corresponding to this physical device consists of 9 bilinear elements and 16 nodes and is presented in Fig. 3.

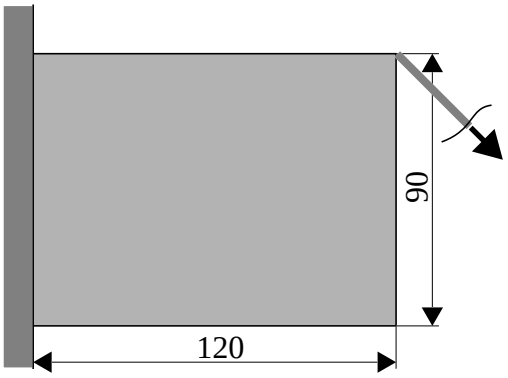

Fig. 2. A simple physical structure consisting of a plate that is exposed to a tension load

- The *extensibility* of AMOSQL provides powerful means for flexible domain modelling. A user/programmer can use very high level declarative query specifications, thus letting the DBMS do data access optimization. AMOSQL also provides extensibility of the query optimizer, which permits an introduction of more complex domain specific cost models that reflect certain aspects of the application domain. For example, solution costs for numerical operations, solution accuracy of numerical calculations, etc., can be included in the cost models. Thus, the query optimization can influence the choice and tuning of operators in FEA.
- Query languages also makes it possible to make advanced *ad hoc queries* concerning the contents of the database. This might be demanded by advanced users and is quite useful since it is impossible to foresee the complete information needs.
- Including a rule system in the query language provides a mechanism for constraint management. Actually, there is an ongoing work aiming at integrating active rules in the AMOS DBMS [27].

The advantages of these features are, of course, varying for different phases in the system or application life-cycle and for different types of application users and system developers. However, the methodology supports an incremental and iterative development, maintenance and evolution of domain models as well as domain modelling systems. It facilitates reuse and evolution of design, including domain conceptualizations and knowledge, by its ability of application and data independent representation.

2.3 Data Modelling and Management in AMOS

AMOS (Active Mediators Object System) [4] is a research prototype of an OR DBMS and a descendant of the WS-IRiS DBMS [21]. WS-IRiS is further a derivative of Iris [28]. AMOS is a main-memory database, i.e. it assumes that the entire database is contained in main-memory and uses disk for backup only. It includes an object-oriented extensible query language, AMOSQL, a derivative of OSQL [6], that is used to model and interface the database. AMOSQL is a functional language, originating from DAPLEX [29] and is also influenced by SQL3 [7]. The data model consists of the basic constructs *objects*, *types*, and *functions*.

The AMOSQL language is also extensible by calling external programming languages like C or LISP, and AMOSQL statements can also be embedded within program procedures.

Objects, Types, and Functions. Concepts in an application domain are represented as *objects*. There are two types of objects in AMOS. *Literal objects*, such as boolean, character string, integer, real, etc., are self identifying. The other type is called *surrogate object* and has unique object identifiers. Surrogate objects represent physical or abstract and external or internal concepts, e.g. mechanical components and assemblies such as

Important research problems in developing domain models for mechanical design and analysis, e.g. using FEA, are to investigate:

- How is the domain modelled using an OO query and modelling language?
- Which domain-oriented data structures are required, and how should they be represented?
- What domain-oriented operators need to be defined?
- How are queries accessing domain-oriented data structures optimized?
- How can AMOS be integrated within a domain-oriented tool, e.g. for FEM analysis?

It is possible to use both programmed procedures and high-level query languages for accessing domain models. A query language is used to define, manipulate and retrieve information in a database. For instance, for retrieving some specific result from an analysis, a query can be formed in the high-level and declarative query language returning the information that satisfies the specified conditions. Combining general query-language constructs with domain-related representations provides a more problem-oriented communication. This approach to data management is more effective compared to the use of conventional programming languages [23][17]. The combination of programming and query languages and their pros and cons for data management is further discussed in Catell [24].

Object-oriented techniques, including object-oriented databases and query languages, are well suited to reduce system complexity. Their applicability are especially suitable to engineering applications which consist of large amounts of complex data and relationships. Specifically, a main-memory object-relational database, like AMOS, is combining high-level modelling with high execution efficiency. The use of declarative object-oriented queries for domain modelling and management offers several advantages over conventional programming:

- *Declarative models* are easier to describe, inspect and understand than procedural programs and thus become more transparent and flexible. A declarative modelling with an object-oriented query-language is compact and (de)composable and makes the domain modelling very flexible and powerful. This problem-oriented modelling approach will naturally produce a representation that is isomorphic to the problem domain.
- Advanced object-oriented query-languages also provide *object views* [25] [26] capabilities. In AMOSQL, views are supported through derived functions, where a function is uniformly invoked independently of whether it represents stored or derived data. This makes it possible to change the underlying physical object representation without altering the access queries. Thus, data independence and evolution are supported.

generation, analysis, and result interpretation. It is further designed in a highly structured, “object-based”, manner with specific sets of procedures for each concept class, such as point, line, etc. This has made it much simpler to integrate it with the AMOS DBMS, since subsets of the domain model can be replaced at a time. The initial integration of TRINITAS and AMOS is described in Sect. 2.4.

2.2 Object-Oriented Databases and Query Languages

Classical DBMS technology concentrated on supporting administrative applications. However, with the advent of OO DBMS there has recently been much DBMS research and development for developing database techniques to support also engineering activities, such as CAD, CAE, and CAM. We distinguish between the *first* and the *next* generation OO DBMS:

The first generation OO DBMS [19] (e.g. ObjectStore, Versant, Gemstone, Ontos) extend an OO programming language (usually C++) with *persistence*, i.e. the possibility to permanently retain C++ data structures on disk. Thus complex engineering data structures (e.g. representing a design) can be built within the C++ programming language by some tool, and then stored on disk. This allows sharing of design data structures between engineers and tools. Access to the database is usually by C++ programming (normally within some tool).

The success of today’s RDBMS is largely due to the availability of query languages that allow non-expert programmers to query and update the database. Query language capabilities are only supported to a limited extent in first generation OO DBMS. The *next generation OO DBMS* [20] or *OR DBMS* [3] (e.g. OpenODB and Montage) also include relationally complete OO query languages to search and update object data structures. The query processors for an OO query language can use most of the techniques developed for relational query processors, but will also need new techniques [21]. Extensibility of the query language will also make it possible to make optimizations over the domain model and thereby domain-related operations [22].

Domain Modelling Using Query Languages. The extensibility of AMOSQL allows the design of *domain models* that represent application oriented models and operators, i.e. FEA models in our case. Domain models allow knowledge and data now hidden in application programs to be extracted from the applications and stored in next generation object databases with domain-specific models and operators. The benefits of using domain models include easier access and management through a query language, better data description (as schemas), and other benefits currently provided only by advanced DBMS such as transaction capabilities and ad hoc query processing. The query processing must be about as efficient as customized main-memory data structure representations to allow the use of local embedded domain models linked into applications without substantial loss of efficiency. Domain models often need to be able to represent specialized data structures for the intended class of applications.

plexity. In the main part of these works, Smalltalk or C++ have been used for implementation.

Compared to OO programming, OR DBMS techniques can extend software design further towards high-level declarative modelling. In the FEA field, database support has so far mainly been used for storage and retrieval of data and results using relational (R)DBMS. However, OO DBMS have also been acknowledged for the engineering field [16], and some of their advantages over RDBMS are reported on in [17]. OR DBMS will probably have an even greater impact on scientific and engineering computing in the future. This paper outlines the opportunities for, and potential impact of, using an extensible OR DBMS for realisation of FEA software.

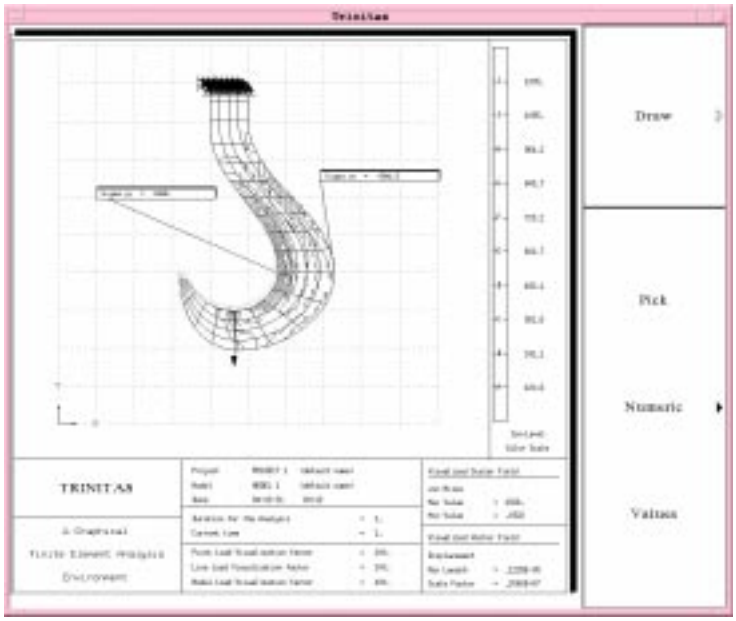

Fig. 1. An example of a FEA model of a hook analysed by a linear static stress analysis

The FEA program used in this work, TRINITAS [5], consists of about 2200 subroutines of FORTRAN code and is positioned somewhere between medium- and large-sized FEA software [18]. It can also be classified as a state-of-the-art FEA program, since it is completely controlled by an interactive graphical user interface and integrates the complete analysis process from modelling to evaluation. This process includes geometry modelling, domain properties definitions, boundary condition definitions, mesh

phasize integration and data management whereas flexible and transparent design might be emphasized by scientific users. The research work is expected to generate insight and experience to both scientific disciplines, i.e. general database technology can be expanded through generic requirements of the application domain (here computational mechanics) as well as the generic scientific database models and systems will extend the field of scientific and engineering computing.

2 Finite Element Analysis Based on Next Generation Object-Oriented DBMS Technology

2.1 Finite Element Analysis and Software

The *finite element method (FEM)* is a general numerical method for solving differential equations. It can be applied to problems within several engineering fields, such as mechanical, civil, and electrical engineering. Within mechanical engineering FEA is used for analyses of designs involving different characteristic design criteria, such as strength, stiffness, stability, resonance, etc. [8]. A typical analysis model includes data of the domain (geometry, material), boundary conditions (forces, prescribed displacements), and of the solution algorithm. The central part of a FEA is the solution of systems of equations of different levels of complexity depending on the phenomenon studied. In the case of a linear static stress analysis, as illustrated in Fig. 1, the central analysis step consists of a solution of a single system of equations. However, even the most basic cases of a FEA involves a large set of data with a high level of complexity. For example the number of unknowns in the equations can range from 100 to several 100 000.

The application of the FEM is mainly achieved by special-purpose programs where the analysis model can be specified in domain-specific terminology. Modern commercial FEA programs have integrated the complete analysis process from modelling to evaluation and also take advantage of graphical user interfaces in this process. However, the basic structure of commercial FEA software and its development have not gone through any dramatic change since its origin, in contrast to the exceptional development in hardware performance during the last 30 years which has been easier to take advantage of by application builders. Conventional FEA software is usually implemented as monolithic programs with various levels of integration of modelling and evaluation phases. The relatively low level of structure in these programs introduces large costs for maintenance, further development, and for communication with other subsystems in the CAE environment. This is motivating research on new software architectures for scientific and engineering computing.

Modern programming techniques, such as OO programming techniques, have also started to influence the area of FEA [9] [10] [11] [12] [13] [14] [15], and has been suggested for design and implementation of FEA software in order to reduce program com-

1 Introduction

The importance of scientific and engineering data management is more and more emphasized in both industrial and scientific communities [1] [2]. This paper describes an ongoing interdisciplinary research work that covers the fields of scientific and engineering database management and computational mechanics. More specifically, we use a *next generation object-oriented database management system (OO DBMS)*, also referred to as *object-relational (OR)* DBMS [3] including a *relationally complete* and *extensible OO query language*, to model and manage *finite element analysis (FEA)*. The discussion in the paper is based on an initial implementation of a system called FEAMOS, which is an integration of a main-memory OR DBMS, AMOS [4], with a FEA program, TRINITAS [5]. AMOSQL, the query language of AMOS, is used to represent and manage the FEA domain model. AMOSQL is a derivative of OSQL [6] but is also influenced by SQL3 [7]. TRINITAS, representing the state-of-the-art within the field of FEA software, completely integrates the entire analysis process and is completely controlled through a graphical user interface. A typical TRINITAS session includes a generation of a finite element model from a specification of geometry, domain properties and boundary conditions, a solution phase, and an evaluation of numerical results. The data representation and its related operators in TRINITAS are piece by piece replaced by a corresponding representation in AMOS. Examples in the paper show typical needs for data modelling and initial performance measures comparing the original FEA software with FEAMOS. Main ideas included are the modelling and management of both structure and process of finite element methodology by the use of an extensible OO query language.

Our idea is to increase the conceptual level in the design of FEA software by using an OO query language to build *domain models* which represent application-oriented conceptual models of data and operators. Class structures and operators are defined to represent finite element (FE) methodology, i.e. FE models and solution algorithms. The extensible query language allows domain-specific FE operators to be included in the system. The user can define queries in terms of the FE model, and the queries may contain FE specific operators. The query optimizer thus needs to understand how to choose among several FE domain-specific operators to answer a query as efficient and as numerically stable as possible. The software developer can take advantage of general and predefined facilities of the OR DBMS for data modelling and management and does not need to write dedicated data structures and access procedures for FEA. Thus, by using declarative modelling techniques through the query language, the complexity of the system can be decreased together with an increased transparency and flexibility.

A database-based FEA application will further facilitate an integration or communication with other parts of the global *computer aided engineering (CAE)* system. It will then be possible to accomplish a global improvement of the efficiency of FEA software from the point of view of the developer, the maintainer, and the user. This will result in an increased life time of the software and an enhanced analysis quality. This has advantages for both industrial and scientific use of the software. Industrial users might em-

Applying Next Generation Object-Oriented DBMS to Finite Element Analysis

KJELL ORSBORN

E-mail: kjeor@ida.liu.se

Abstract

Scientific and engineering database applications put new requirements on database management systems that is usually not associated with traditional administrative database applications. These new database applications include *finite element analysis (FEA)* for *computational mechanics* and usually have a high level of complexity of both data and algorithms, as well as high volume of data and high requirements on execution efficiency. This paper shows how *next generation object-oriented database technology* that includes a *relationally complete* and *extensible object-oriented query language* can be used to model and manage FEA. The technology allows the design of *domain models* that represent application-oriented conceptual models of data and operators. An initial integration of a *main-memory object-relational database management system* with a state-of-the-art FEA program is presented. The FEA program integrates the complete FEA process and is controlled completely through a graphical user interface. Examples are included of the conceptual model and its manipulation along with some initial performance measures. It is shown that the integrated system provides competitive performance and is a promising alternative for design and implementation of future FEA software.

This work has been supported by
The Swedish National Board for Industrial and Technical Development

Published in
Witold Litwin, Tore Risch(eds.): Application of Databases,
1st Intl. Conf., ADB-94, Vadstena, Sweden, June 21-23, 1994 (Proceedings),
Lecture Notes in Computer Science, Springer Verlag, ISBN 3-540-58183-9, 1994.

IDA Technical Report 1994
LiTH-IDA-R- 94-16
ISSN-0281-4250

Department of Computer and
Information Science
Linköping University
S-581 83 Linköping, Sweden