Computer Assisted Mechanics and Engineering Sciences, 2: 271–287, 1995.

Object oriented software methodology: a case study of systems for dam monitoring, state diagnosis and visualisation

Aleksander Radomski and Olaf Gajl
Institute of Fundamental Technological Research, Polish Academy of Sciences
ul. Świętokrzyska 21, 00-049 Warsaw, Poland

(Received January 30, 1995)

The complexity of the physical engineering objects requires new technologies in software development able to simulate real-life cases. The huge number of such cases can be covered by object-oriented paradigm. This general idea and some advantages of using object-oriented language (Smalltalk) are exemplified by a presentation of a system for earth dam control. The system is an expert type program equipped with advanced monitoring and visualisation functions for existing dams. The software development process starting from the requirement description is presented. The structure of the dam model and of the inference engine as well as of the class hierarchy is shown as the examples. The re-usability of the system is proved by its implementation for different earth dams.

1. INTRODUCTION

For several years a visible difference between the rate of development of software and hardware could have been observed — the progress in software was markedly lagging behind the increased abilities of hardware. This situation resulted in development of the branch of computer science called software engineering. Since some time, many software engineers claim that only the more widespread use of methodology based on the paradigm of object-oriented programming (e.g. [7, 14]) opens perspectives for qualitative changes in the rate of development of usable software. Practical applications in technology need software that takes into consideration the great complexity of the problems described, as well as the limitations on the side of the software user. These requirements are difficult to satisfy and combining them demands a wide application of graphical user's interfaces, together with the methods of artificial intelligence (including expert systems). On the other hand, to simulate behaviour of complex engineering objects, an application of the methods of qualitative analysis seems necessary, in addition to the classical numerical methods (see [16, 17]). However, irrespective of the need to carry such basic research, it is necessary to use the present state of knowledge in practical applications — which was the aim of the work that led to this paper.

The complexity of problems, the multitude of disciplines describing real engineering structures, and the need to provide an easy communication with the user result in high costs of building the software. They constitute the most important factors limiting the development of engineering software. In classical applications, the amount of labour needed to build the user interface exceeds about five times the amount needed to build the calculation part proper. An additional problem is the difficulty to adapt the system when the needs of the user change.

The methodology based on the object-oriented programming paradigm promises to break the cost barrier, facilitating easy modification, and speeding up the development of both the graphical user interface and other parts of the system. In the paper we present an application of this methodology to systems supporting exploitation of earth dams.

The phase of building the system presented in this paper has also been considerably shortened,

even though the system has a much broader scope than the classical expert system. On the other hand, the phase of knowledge acquisition has not been shortened. There is a number of tools supporting the acquisition of knowledge (e.g. [4, 5, 9]), which may shorten the process of building the knowledge base. However, this aspect of the problem has not been considered in this paper.

The expert systems may be categorised as *analytic* or *synthetic* [12], or else — using the terminology of [1] — solving the engineering tasks of the *derivation* or *formation* type. The first group contains *diagnosis* (searching for causes of defect or malfunction), *interpretation* (searching for models in collections of facts, readings, etc.), *monitoring* (watching changes in the system in order to interfere in the case of emergency), and *prediction.* The other group contains tasks such as *design, planning* and *assembly.* The system presented in the paper is a typical representative of the first group, although most of the considered aspects may also be applied when building a system supporting, e.g., a design task.

Most of the expert systems addressing problems connected with exploitation of hydrotechnical objects refer to concrete dams [11] or retaining walls [6]. Because of the specific character of those objects the analysis of their behaviour and possibility of failure has a different character than in the case of earth dams. The complexities of mechanical phenomena together with occurrence of filtration effects are the reasons for a different instrumentation of analysis [20, 3]. In the systems described in the paper the calculation modules have not been included — usage of finite elements method analysis is placed outside of the system and serves to determine the border values of parameters describing the state of the dam. Yet, including the finite elements method is certainly possible [8]; there is a number of object-oriented programs available for that task (e.g. [2, 19]).

2. Modelling real-life engineering objects

The basic problem to be solved by the methodology of creating software modelling the engineering aspects of real-life objects is the complexity of the task. It is due to the complexity of the field itself as well as to the specific demands made by the users of such software systems.

2.1. The domain

A multitude of formal disciplines participates in modelling real-life engineering objects, each describing in its particular way the physical phenomena relevant to the given class of objects. In each of them the methodology of modelling is burdened with great inborn complexity, each introduces a separate terminology, physical theories and appropriate mathematical language. The term "real-life objects" is supposed to mean that for a sufficiently adequate presentation of all aspects considered essential from the engineering point of view, it is impossible to apply assumptions simplifying or limiting the model to only some selected aspects, contrary to the common academic practice. It results in an essentially interdisciplinary character of production teams whose members cannot avoid the difficulties of co-ordination of terminology and the effort of crossing borders of their own disciplines. This cognitive and managerial difficulties were particularly early recognised in the case of artificial intelligence (expert) systems; in the literature (e.g. [5]) it is stressed that a serious problem here is the so-called *acquisition of domain knowledge* from experts who usually have no experience of co-operation in making computer systems.

The common present-day approach to computer-supported modelling of engineering problems consists of integrating existing software packages which were made with traditional technology, within the limits of narrow paradigms, e.g. CAD systems, structural modelling systems (especially those based on finite elements method), the systems registering measurement data, and systems of visualisation of data and results. In the case of human teams, problems of communication arise mostly at the level of terminology, whereas in the case of specialised software packages they arise at the level of translation of data formats and organisation of co-operation of different subsystems of the target system.

2.2. The user's needs

It is obvious that the structure of needs of the user of such systems results from the structure of the underlying engineering problem. Yet, ideas concerning the scope of computer assistance to the engineer change, along with the increase of possibilities to apply hardware and software to tasks which were so far performed "manually". With increasing accessibility of more and more powerful computing equipment, we observe a growing demand for multiaspect systems, aiding all phases of modelling, analysis and presentation of results.

Another measure of usability of system functions is the simplicity of tools offered by it and their adequacy to the real needs generated by the problem. The present state of affairs is that on the one hand we have commercial "all purpose" packages (like ABAQUS, Mathematica, AVS, FemView), which are not very flexible and, although they offer hundreds of functions attested by thousand-page manuals, when facing an actual problem they often are by themselves not sufficient to model it adequately. On the other hand, software systems appear which are made *ad hoc* to solve some particular problem — they, in turn, often are not universal enough and hard to apply to different, though formally related problems.

Finally, attention should be given to the possibility of using the systems in many programming environments (hardware and software platforms). If the software system is not a fully integrated package but is made of many different tools, transferring it to another environment, if possible at all, calls for expert assistance of a team of computer science professionals.

The problems mentioned above were so far not systematically analysed, although in many papers the need for undertaking the task has been indicated.

2.3. Traditional versus postulated software engineering methodology

The paper attempts to show an alternative to the currently predominating methodology of production and maintenance of computer software systems designated to modelling of engineering tasks.

The commonly applied technology (called here "traditional") can be characterised as follows. The process of constructing the system engages for a long time large teams of experts in particular aspects of the modelled object, computer science, ergonomics and man-machine interfaces. The complexity of the problem and incompatible means of description lead unavoidably to difficulties in formulating the specification and to incoherence of the resulting system.

A side effect of this procedure is a limited flexibility of the product and its resistance to modifications of its specification (in regard to both the scope of functions offered by the system and the maintenance tools). Moreover, because of very high costs of production and servicing, it is unprofitable to produce a professional systems dedicated for solving a narrow range of problems in a specific way, i.e. systems in which only narrow groups of users are interested. That is why today the tendency prevails to form "all purpose" packages (aimed for everybody but in fact for nobody). It is assumed that the final application will be configured by the end user himself using the tools offered by the package. However, it is a task which can be successfully performed only by an expert trained in using the package. At the base of that situation lies also an economic mechanism: the user of commercial package pays for the capabilities which in most part he will not use (and which he does not need), whereas the manufacturers of (more and more powerful) hardware are interested in a software market where over-developed systems demanding yet more powerful hardware prevail (and so the circle is closed).

This analysis of commercial software market points to a certain inadequacy of the traditional technology for the user needs, particularly concerning systems dedicated to narrow groups of users. Such diagnosis, although incomplete, makes it possible to list the most essential postulates to be fulfilled by the alternative technology:

- It should be based on the programming paradigm that enables a uniform formulation of all aspects of the system, reducing the need for keeping large teams of narrowly specialised experts.
- The programming environment should enable rapid realisation of a prototype system (with respect to the scope of functions and the ways of maintenance) and provide tools for easy, dynamical introduction of modifications of specifications. The end users of the system (who will use it as a tool for modelling of engineering tasks) should be able to influence its shape and functionality from the earliest stage of design.
- A system dedicated to a narrow class of problems should still be designed with consideration to its reusability (e.g., with a division of the software into usability levels: upper levels being common to a broad class of problems, and lower ones — specialised in generating specific solutions for more narrow classes), as it is economically justified to form "open systems" (such that their modules may be used again in other applications and could be easily preserved and modified).
- Of substantial importance is the property of self-documentation of the system (well thought out, not *ad hoc* user interface, resulting from the structure of the problem, not from the structure of available programming tools); using the system cannot demand a thorough study of many hundreds of manual pages.

According to our opinion, the technology with such features is offered by the programming environments based on the object-oriented programming methodology, a paradigm more and more approved both by the producers of commercial software and the academic circles dealing with software engineering (see e.g. [10, 18]). In the subsequent part of the paper we will show a practical application of such a methodology, using the example of a series of monitoring systems for earth dams, designed and implemented in our Institute over the past three years.

3. A SYSTEM FOR MONITORING, DIAGNOSIS AND VISUALISATION OF EARTH DAM STATE

The task consisted in constructing a computer system which would assist the personnel of an earth dam in monitoring its state and diagnosing incorrectness of its functioning. The system should also archive and present the data collected during the exploitation of the dam. The system should have the following functionality:

- registration of measurement data from the control and measurement apparatus (pressure cells, piezometers, and others);
- registration of observations of the state of the object done by the personnel;
- rule-based inference for diagnosis of the state of the object and signalling possible hazards;
- visualisation of current and past (archived) data — for purposes of documentation and periodic analyses.

At the beginning, it was assumed that the system will be aimed at a very narrow group of users connected with a single real object — the earth dam built at Czorsztyn (Poland). An adaptation of the system for similar dams, in the case of success, was also considered.

3.1. Conditions of design

Such formulation of the task determined the following design conditions:

- the first aim was to produce a workable system in a short time (due to the advancing construction of the dam at Czorsztyn);

- the expected (and proved in practice) difficulties in knowledge acquisition from experts implied a choice of system architecture with a simple knowledge representation scheme, enabling its fast implementation;
- a possibility of adaptation of solutions to similar objects (in this case: other earth dams) had to be taken into account — hence the need for minimalization of procedural specifications (encapsulated in the system) in favour of exchangeable and easily modifiable declarative specifications;
- strong emphasis had to be put on easy interaction with the users of the system, as most of them practically never got in contact with computer systems before;
- the small size of the construction team required the use of effective methodology of design and implementation of the system.

3.2. Sources and representation of knowledge

The input data for the system were the quantitative readings of the control and measurement apparatus (gauges) and the qualitative evaluation of the state of particular structural elements of the dam. For the subsystem supporting the diagnosis of the state of the dam it was assumed that the rules would be formulated in the qualitative form (using such terms as *"readings normal"* or *"fast increase of readings"*). The knowledge formulated in this way has a more general character (i.e., it is applicable to broader class of objects).

The quantitative readings from the gauges had to be transformed into a qualitative form. The quantification procedures should be based on domain-specific criteria of quantification (coming from simulation and calculation models of earth mechanics or filtration phenomena). However, the verification of correctness of these quantification criteria had been considered to lie outside of the system.

Thus, the sources of knowledge and methods of its representation were divided into three levels — general qualitative knowledge (rules), quantification procedures specific for the object (the dam), and quantification criteria (external in relation to the system).

3.3. The user, or the functional scope of the system

For a system of this type it is also necessary to define the term *"user"*, in a way corresponding to the functional scope of the system. Thus we distinguished:

- the *"everyday user"*, i.e. the maintenance personnel of the dam: an interface for input of observations from inspection tour of the object is necessary (registration of measurement data is partly automatic); the user expects tools for visualisation of measurement data and access to the data from the history of the object;
- the *domain expert*, i.e. an earth dam specialist evaluating the state of the dam at longer intervals (e.g., every six months): he must have access to all collected measurement data and selective access to the data base containing the history of the dam (e.g., a list of all malfunctions noticed for the given region of the dam);
- the *knowledge-engineering expert*, i.e. a person verifying adequacy of the knowledge base (rules) and quantification procedures: he evaluates the functioning of the system over a long period of time, taking decisions about a probable changes of the inference rules and quantification criteria; he requires the data necessary for, e.g., an additional verification of the methods used for modelling, and an access to the modules storing the rules and quantification procedures.

3.4. The implementor, or the range of necessary programming techniques

From the software engineering point of view, design and implementation of the system required an application of techniques from the following disciplines:

- *expert systems* (acquisition and representation of knowledge; implementation of inference engine; justification of diagnosis results);
- *data bases* (archiving quantitative and qualitative facts describing the history of the object; generating reports);
- *computer graphics* (visualisation of data: charts displaying quantitative data, tree diagrams presenting justification of the diagnosis);
- *user interface design*, taking into account different types of users with different needs (ergonomy, organisation of interaction, visual design).

4. Realisation of the task

Below we shortly present the process of realisation of the above characterised task. The course of presentation corresponds generally to system design sequence.

4.1. Choosing the paradigm of model formulation: object-oriented programming

Object-oriented programming is characterised by the features which, in our opinion, well correspond to the needs discussed above. These features include *abstract data structures*, strictly *separated code*, and *inheritance* (generalisation and specialisation facilitating modularisation and reusability of the software). The possibility of expressing aggregated and well-structured data types enables to write compact and readable programs. The paradigm also enables a conceptually (and formally) uniform notation of various system modules.

4.2. A note on programming environment used

In our case it was Smalltalk-80 language [13] with the VisualWorks™ programming environment (from ParcPlace Systems). As it is not a commonly known tool, the choice needs a justification. Among the people who are in everyday contact with current software engineering practice it is rather common to reduce object-oriented programming to the popular programming language C++. We appreciate its popularity but we must state that for a language aimed at real-life object modelling, C++ is too engrossed with details of purely technical character. The latest realisations (e.g., Microsoft Visual C++ ver. 1.5) are also subject to the "rule of large commercial packages", as formulated above. The Smalltalk environment is easily portable to various software/hardware platforms like UNIX (with X-Windows), Microsoft Windows, Macintosh and others, and after installation takes only 8 MB of hard disk space, whereas Visual C++, besides taking over 150 MB of disk space, enables construction of applications which may run only in the MS Windows environment, and cannot be mastered — as a tool — in a period of time shorter than a few months. Thus it is not a language which we would recommend for use by small teams which want to produce a professional engineering application cheaply and fast.

4.3. Internal system structure

According to the object-oriented programming methodology, the system was built around the data structures representing both the elements of the real object and the functional elements of the system itself. They are:

- *The dam model* — a declarative data structure (formally: a hierarchy of descriptions, structured as a tree), reflecting the division of the dam into subobjects. The model stores topographical, quantitative and qualitative data pertinent to the given level of description. Also the data base storing the history of the dam is organised around the model. The model represents the facts.
- *The inference graph* — a declarative data structure representing the deduction rules (formally: an oriented graph, with edge direction representing the direction of implication). The nodes of the graph correspond to facts (hypotheses, confirmed conclusions), formulated in qualitative terms.

From the dynamic point of view, the operation of the system is structured according to the cycle: *fact — quantification — deduction — result of deduction (fact or additional question)*. The information flow and internal structure of the system are depicted in Fig. 1.

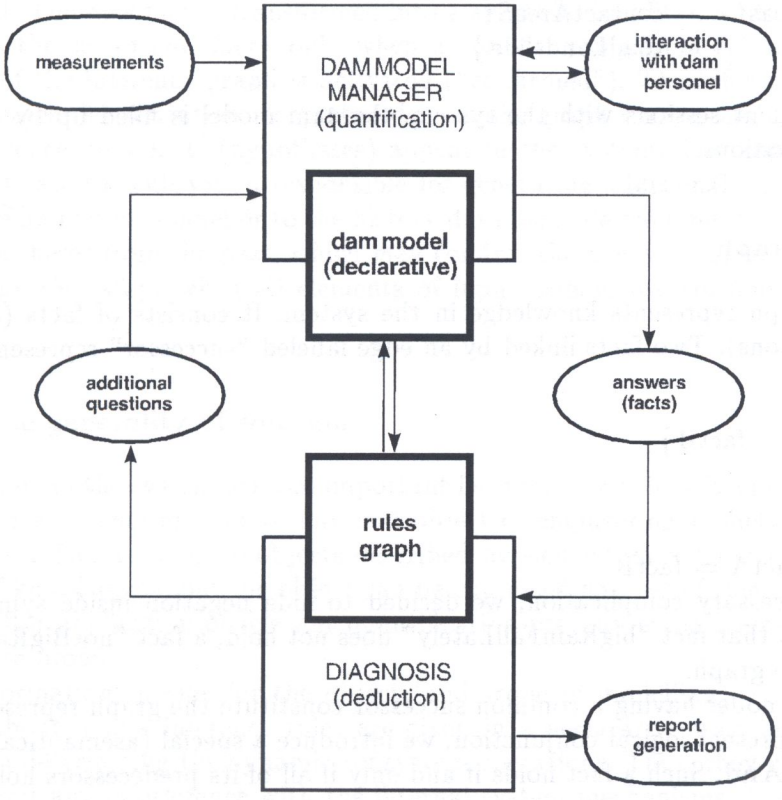

Fig. 1. Internal system structure

4.4. Dam model

The dam model is defined declaratively by means of a set of structural objects. A structural object is a collection of pairs: *attribute name/attribute value*, where attribute values include also links (or sets of links) to other structural objects. Two examples of structural objects describing parts of a dam are given below. **DAMPart** and **ObservationZone** are class names (in Smalltalk sense).

DAMPart damModel
{ zones: (observationZone1 observationZone2 observationZone3
observationZone4 observationZone5 controlGallery)
upWaterLevel: upWaterLevel
downWaterLevel: downWaterLevel
rainFall: rainFall
maxWaterLevel: 398.60
crestLevel: 400.20 }

ObservationZone observationZone1
{ lex: #observationZone
partOf: damModel
id: '1'
fromHm: 0.0
toHm: 130.0
sections: crossSection1
crest: crestS1
downstreamSlope: downstreamSlopeS1
upstreamSlope: upstreamSlopeS1
contactAreas: contactAreaS1
landslides: roadLandslide }

During subsequent sessions with the system, the dam model is filled up by data coming from gauges and observations.

4.5. Inference graph

The inference graph represents knowledge in the system. It consists of facts (structural objects denoting prepositions). Two facts linked by an edge labeled "successor" represent an implication:

Fact factA
{ succ: factB }
Fact factB
{ ... }

is equivalent to: factA $\Rightarrow$ factB.

To avoid unnecessary complication, we decided to hide negation inside symbolic fact names. Thus, to represent that fact "bigRainFallLately" does not hold, a fact "notBigRainFallLately" has to be added to the graph.

Two (or more) nodes having a common successor constitute the graph representation of logical disjunction. To represent logical conjunction, we introduce a special (asemantical) kind of facts — instances of class **And**. Such a fact holds if and only if all of its predecessors hold:

Fact factA
{ succ: and1 }
Fact factB
{ succ: and1 }
And and1
{ succ: factC }
Fact factC
{ ... }

is equivalent to: factA & factB $\Rightarrow$ factC.

Fact "holds" when it is confirmed. Fact can be confirmed by application of the above described rules to the inference graph; leaf nodes of the graph (nodes without predecessors) are confirmed by quantification procedures. Thus, if any of automatic piezometers located in the measurement section 1 exhibits fast increase of its reading, a fact "piezometerMeasurementSection1FastIncrease" will be confirmed. Inference mechanism then tries to propagate the "confirmed" state along edges of the inference graph. The upper bound of the confirmed facts (i.e., a subset of confirmed facts which have no confirmed successors) is taken as a result of the inference.

Such an inference scheme corresponds to a production system without variables and with forward chaining using depth-first search. Formally it is an approach based on a finite-state automaton, rather than the "classical" one widely used in expert systems.

The important feature of our approach is its opportunistic character. Facts (graph nodes) represent malfunctions (states of abnormal functionality of the dam). Thus, the inference mechanism is activated only when gauge readings or observations recognised as malfunction signals are registered.

4.6. Control flow scheme

Input stream for the system consist of measurement data readings and qualitative evaluations introduced into the system by the personnel of the dam. In the first case the quantification procedures are activated. The new fact is transformed into a symbolic (qualitative) form. The inference subsystem reviews the set of new facts: only when a fact deviating from the norm appears, the appropriate node of the inference graph is activated ("confirmed"). The inference subsystem automatically propagates the activation along the edges of the graph corresponding to implications. As a result of inference, new facts (hypotheses) appear in the system. The appearance of a new hypothesis may activate the subsystem responsible for generating additional questions. They may be addressed to the human personnel or to the history data base (demanding, for example, retrieval of the facts or hypotheses from the past, which may confirm the newly generated hypothesis).

The diagnosis process stops when all elements of input stream are consumed in the manner described above.

4.7. A note on the generality of solution

The internal structure of the system has two important features. First, it is formulated on a general, abstract level. Such a scheme may fit to any real object of engineering modelling, if only it can be decomposed into a hierarchy of subobjects described by elementary attributes (quantitative or qualitative), and if knowledge about the object can be expressed as a set of rules formulated at the qualitative level, after application of relevant quantification procedures. It seems that the class of such objects is quite broad.

Second, the information specific for the object (and scope of modelling), i.e. the structure of the object, facts about it and inference rules are given in a declarative way, independent of the internal mechanism determining the dynamics of system operations. Thus, they can be replaced by another data without any interference with the internal system mechanisms. It does not apply to quantification procedures — in the present version of the system, they are given at the procedural level and must be formulated anew for another real object. Extension of the system by a module solving (declaratively given) equations and constrains that determine the quantification rules should eliminate this difficulty in the future.

4.8. Functional structure of the system

By the functional structure of the system we understand the range of functions offered to the user and tools of interaction by means of which the user can activate these functions. The functional

structure should follow from the structure of user's needs, in the same way as the internal structure of the system is built around the data structures representing the model of a real object.

The process of interface design was interactive: we started from a prototype (in which, of course, the internal subsystems were not completely implemented), and the recipient of the product (who represented various final users of the system, see Section 3.3) reviewed its functional structure and proposed modifications and additions. After some time (usually a few weeks) the cycle was repeated. It should be added that the process was running in parallel with the implementation of the internal mechanisms of the system as well as with the formulation and refinement of the knowledge base rules (knowledge acquisition from experts).

The system presents its functionality on the computer screen with three groups of windows. They are:

- *inspection tour questionnaires* (filled everyday by the personnel of the dam),
- *dam views*: planar (topographic) view of the dam, the measurement cross-section views and individual gauge views,
- *system reports*, either popping up automatically (e.g. signalling malfunctions of the dam as detected by the diagnosis subsystem), or generated on the user request (e.g. excerpts from the history database, charts of quantitative measurement data).

Functional scheme of the system windows, i.e. how the user can pass from one window to another, is presented in Fig. 2; examples of various kinds of windows are given in colour plates (Figs. I–VI).

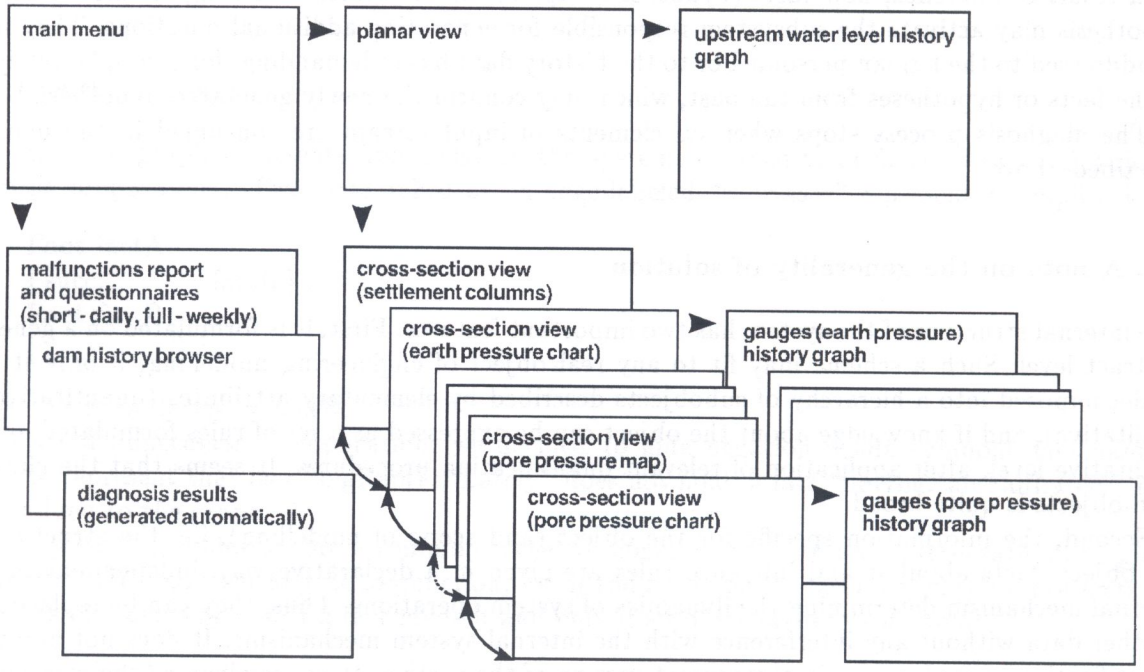

Fig. 2. Functional scheme of the system windows

5. Design details of selected modules

A description of all design and implementation details of the system lies outside the scope of this paper. However, we want to illustrate the adopted methodology of object-oriented design and programming with description of solutions to selected design problems.

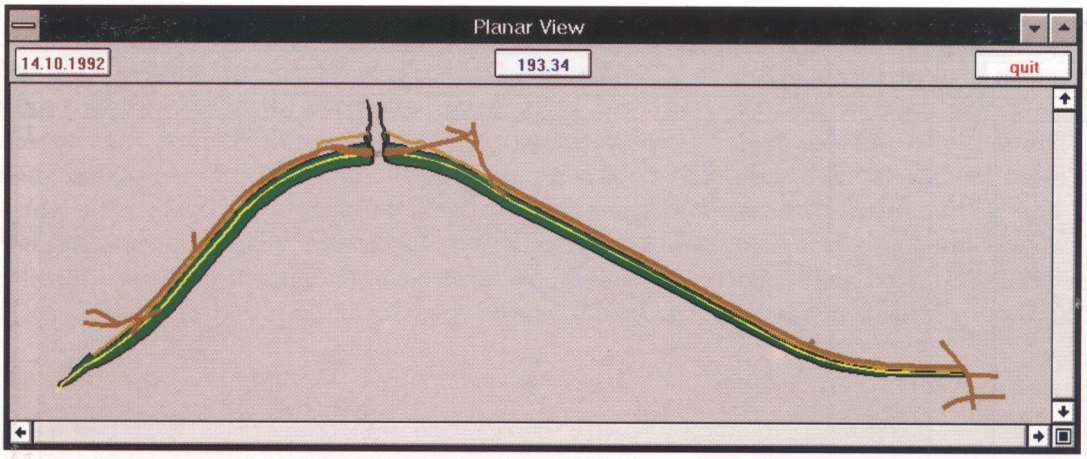

Fig. I. Window presenting landscape view of NYSA dam

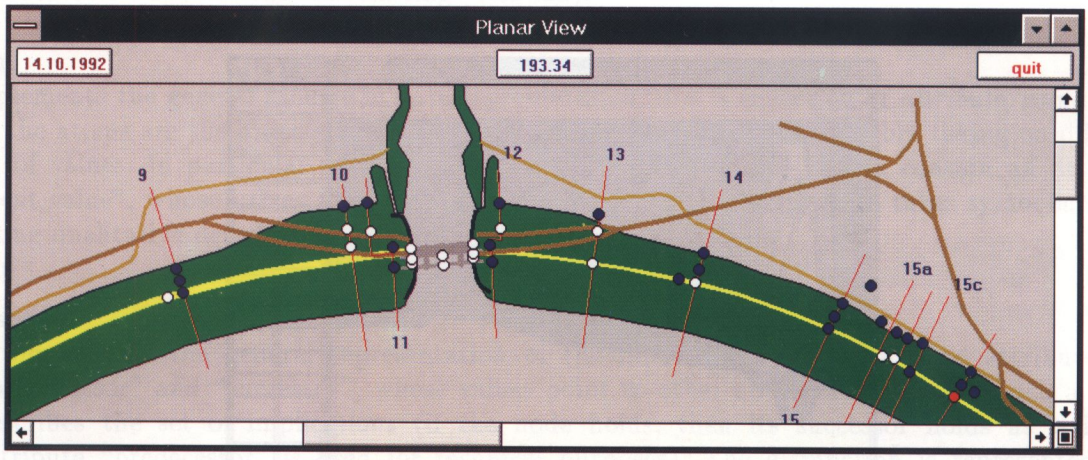

Fig. II. The same window after zoom

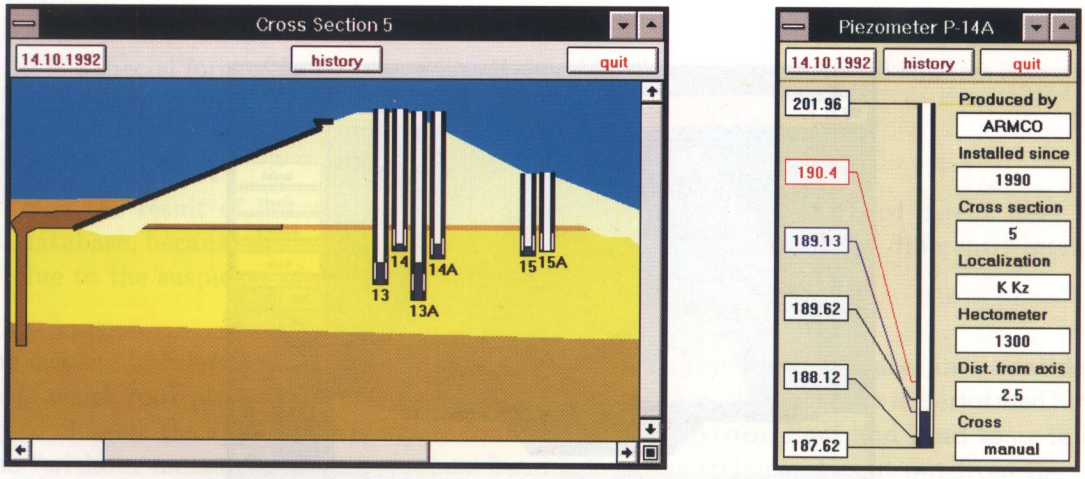

Fig. III. Window presenting cross section view (left) and open piezometer data (right); both taken from system for monitoring of NYSA dam

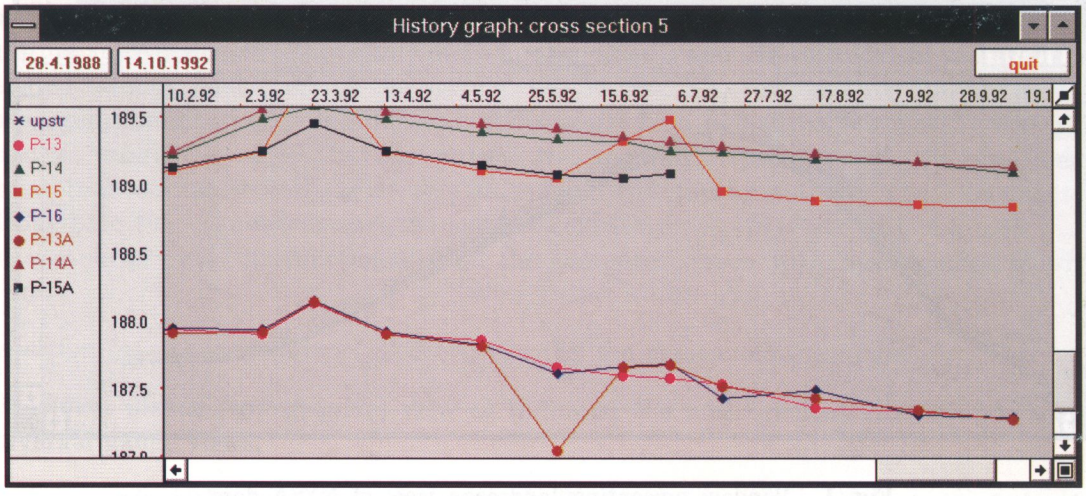

Fig. IV. Chart of open piezometers' values (NYSA dam)

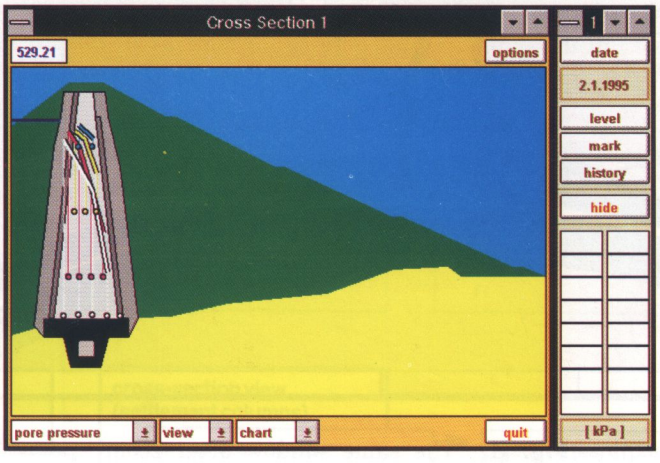

Fig. V. Cross section view with accompanying options window (CZORSZTYN dam)

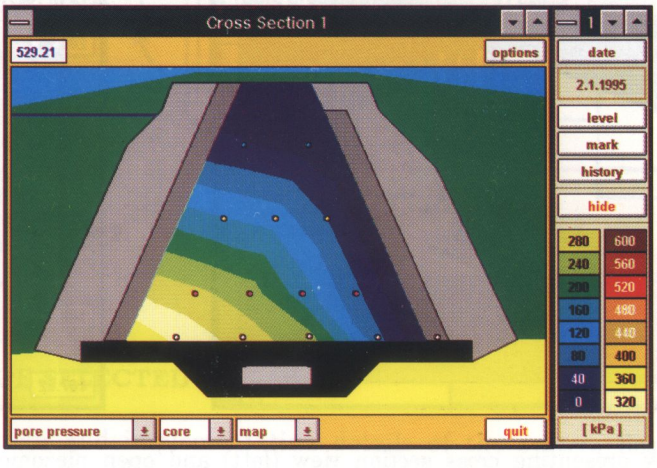

Fig. VI. The same windows with another presentation option turned on

5.1. Program layers

One of the major software engineering postulates is to produce software in regular and modular way. Within the object-oriented programming paradigm that postulate is realised through formation of program layers, of which the upper ones describe the most abstract aspect of a given program element, and the lower ones realise the specialised particulars. The object-oriented programming languages, including Smalltalk-80, supply tools to express such stratification. They include *classes* (to describe objects of identical internal structure and functionality), and *inheritance* mechanism (to specify functional specialisation). These tools are so general that they can be applied to seemingly different aspects of the software system, like the internal representation of data and the mechanisms of user interaction. This can be illustrated by two examples below.

5.2. Atoms, facts, hypotheses

For the needs of the deductive diagnostic subsystem the following structure of classes was created (the indentation denotes the "subclassing" relation).

Atom
It implements the general scheme of data representation: it is a collection of *attribute/value* pairs. Thus the atoms are the objects describing arbitrary sets of attributes, possibly taking on different types of values. In particular, such elements of the dam model as gauges contain an attribute "current value", whose values are numbers, and an attribute "state" which takes symbolic values denoting qualitative descriptors.

Fact
It implements the general representation of the inference graph node. The attributes are "predecessor" and "successor", whose values point to other nodes of the graph. Such a graph describes the set of implications (if the node holds, then its successor holds as well; the attribute "predecessor" is used for technical purposes). The disjunction is represented in a natural way: if a given node has some predecessors, it is confirmed when any of them has been confirmed.

And
It is a special form of fact representing the conjunction. Such a node will confirm its successors if and only if *all* its predecessors are confirmed.

FactFromPast
It is the result of inference done in the past. Such results are stored in the dam history database, because in the future a fact may occur which will have to be differently interpreted due to the suspicion of possible malfunction inferred in the past.

The scheme of inference used in the systems of earth dam monitoring takes into consideration the facts which took place in the past (before the moment when the inference is performed). Still, it is easy to imagine the application of layers containing classes **Atom**, **Fact** and **And** to an inference scheme "without memory": the appropriate fragments of the code can be cut out from the system and used in another application practically without modification.

The knowledge base consisting of the inference graph will be discussed, from another point of view, in Section 6, in connection with knowledge acquisition.

5.3. Views of the world, dam, and dam cross-sections

An important element of the user interaction with system are the views through which he can access the elements of the dam (represented in the model) which interest him at the moment, down to individual gauges. In the system, a hierarchy of classes supporting such interaction was implemented.

WorldView
This class implements scaleable diagrams, keeping proportions when enlarged or contracted. The objects of this class can also perform one of four operations specified by a mouse button: *select, zoom in, zoom out* and *pan*. At this level only the last three are implemented (as they are independent of the contents of the diagram). The operation *select* is implemented in subclasses (as it depends on the particular type of displayed diagram).

DamView
It is a general representation of views of different aspects of the dam, with a possibility of interaction specified by mouse movements and buttons. On this level, the idea of a diagram element reacting to the *select* operation appears. Still, the realisation of this operation requires further specialisation in subclasses.

LandscapeView
It is a topographic (planar) map of the dam with a possibility for interactive opening of measurement cross-section views and windows presenting collected information for particular gauges. In this view, the symbols of cross-sections (lines) and the symbols of gauges (small circles) react to the *select* operation.

CrossSectionView
It displays a measurement cross-section of the dam with a possibility for interactive opening of windows describing particular gauges. The symbols of gauges (e.g., an open piezometer, represented by a partially filled thin rectangle, with degree of filling corresponding to current value of the piezometer) react to the *select* operation.

The design of graphic display of views capable of proportion-preserving scale change (i.e. the functional contents of the **WorldView** layer) is not a trivial task. This layer should work properly in different systems used in entirely different application areas. As an example, we may consider a system displaying the shortest path between two points on a town map at the user's request (which requires quite another specialisation of the *select* operation).

6. Knowledge acquisition

The central problem of expert system construction is the problem of acquisition and formulation of knowledge. The literature on the subject [5, 12] clearly demonstrates that knowledge acquisition constitutes the most difficult phase of design. A general methodology of interviewing experts has not yet been elaborated, although a number of recommendations or general methods of procedure are proposed [9]. However, it is difficult to apply algorithmic approach to the process of knowledge formulation, especially as at the very beginning an essential difficulty appears — that of fixing common concepts and terminology among system implementors and domain experts.

In our system, these difficulties were solved step by step. The first phase was an attempt to compile a vocabulary of terms used. It was built in a way natural for the domain experts, according to a hierarchically structured description of the engineering object, in our case a dam. The successively introduced terms, denoting elements of dam construction and physical processes occurring both in normal functioning and in malfunction, were precisely defined. The static part of the vocabulary was transformed into a declarative dam model. Subsequent subclasses of the class

DamPart (a subclass of **Atom**) were formed, such as **CrossSection**, **Gauge**, etc. Then, first diagnostic rules were formulated, based on the knowledge of physical processes going on within the dam. At the beginning, they were formulated as phrases in the natural language, in the implicative form ("if... then..."). In this phase of work, the experts concentrated on the essence of the physical processes and consequences of anomalies detected by the control and measurement apparatus, paying less attention to precise formulation and formal correctness of the rules created. The number of terms with inaccurate meaning increased, which led to the compilation of another vocabulary — of the terms used in rules. Later on, the rules were transformed into a declarative notation using instances of classes **Fact**, **And** and **FactFromPast**.

Successively, with the increase of the number of rules, the original form of the notation lost its value. It became evident that representing the rules graphically, in the form of a "rules graph", is more clear and consistent. After a period of adaptation, use of this graphical notation allowed the experts to formulate a much subtler interdependencies between the premises and conclusions. Soon the number of rules increased over 100.

The further step, according to the majority of known approaches, would have been a linearization (into a sequence of textual rules) of the final rules graph, according to some adopted sequencing convention. However, within the object-oriented software methodology described in this report, it is enough to modify the definitions of the objects of subclasses of **Fact** by appropriately setting values of the attribute "successor". Completing the graph by setting values of the attribute "predecessor" was implemented as a (trivial) procedure in the system. It also turned up that it is easy to add the independently implemented data visualisation module [17] to the development environment of the system. Among other things, the module was able to visualise trees (hierarchies). Thus, we obtained immediately a tool for visualisation of the inference graph (see Fig. 3), which also enables fast verification of the formal correctness of introduced rules.

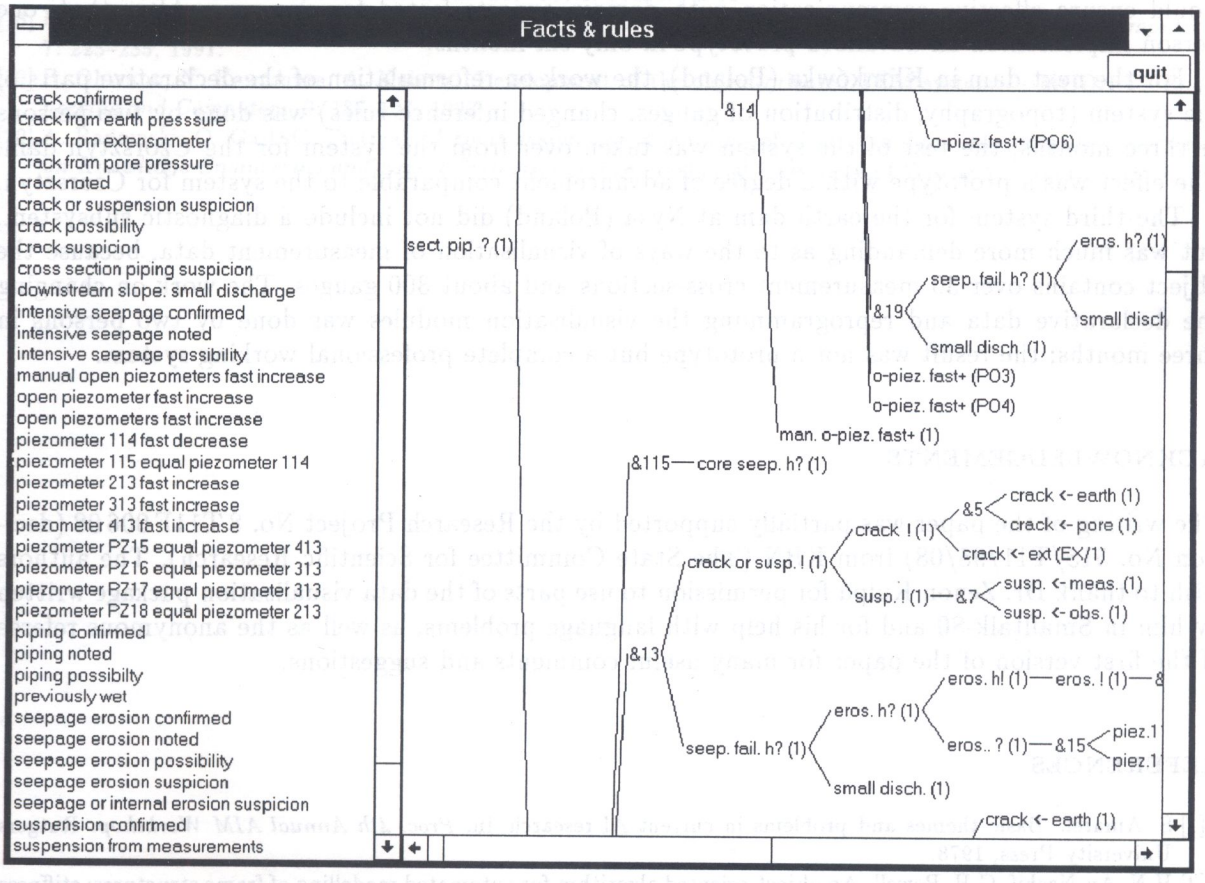

Fig. 3. Facts and rules graph browser

7. Conclusions

The paper discusses usability of traditional technology of software production, as opposed to the modern object-oriented software methodology, trying to demonstrate that in many contexts the former one cannot be effectively applied, particularly in the case of dedicated systems for a small group of users. One example of such a case is the problem of modelling of real-life engineering objects, illustrated in the paper by a system for dam monitoring, state diagnosis and state visualisation, as implemented by the team including the authors.

Within the object-oriented programming paradigm, especially with the support of an effective software development environment (like VisualWorks™ used by the authors), it is possible to prepare a prototype of the system very fast, and to confront it with the end user before too many decisions about the structure and functionality of the system become irrevocably fixed. It also allows to overcome the serious psychological barrier following from a necessity to begin the work on the system from scratch.

Another important effect, offered by the object-oriented technology, is the possibility to achieve the substantial aim of the software producers, namely a possibility of real software reusability. Thus, if proper care is taken to consider that requirement during system design, the work on each new system dedicated to a similar field or type of modelled objects will not be starting from scratch, but from still higher and higher development level.

Finally, we would like to present shortly the history of construction, in our Institute, of three systems dedicated to earth dams, as it seems to confirm the above observations. In fact, the experience with construction of these systems inspired us to formulate these observations in the first place.

In the case of the Czorsztyn dam the work was started by a team of three persons. The preparatory study and research on the formulation of such a scheme of knowledge representation which would ensure effective communication with domain experts lasted for one year. After that, one person implemented an advanced prototype in only six months.

For the next dam in Klimkówka (Poland), the work on reformulation of the declarative parts of the system (topography, distribution of gauges, changed inference rules) was done by two persons in three months; the rest of the system was taken over from the system for the Czorsztyn dam. The effect was a prototype with a degree of advancement comparable to the system for Czorsztyn.

The third system for the earth dam at Nysa (Poland) did not include a diagnostic subsystem, but was much more demanding as to the ways of visualisation of measurement data, because the object contains over 30 measurement cross-sections and about 300 gauges. The work on changing the declarative data and reprogramming the visualisation modules was done by two persons in three months; the result was not a prototype but a complete professional working system.

Acknowledgements

The writing of the paper was partially supported by the Research Project No. 8 T11F 006 08 (decision No. 515/T11/95/08) from KBN (The State Committee for Scientific Research). The authors wish to thank Dr. Zenon Kulpa for permission to use parts of the data visualisation package written by him in Smalltalk-80 and for his help with language problems, as well as the anonymous referee of the first version of the paper for many useful comments and suggestions.

References

[1] S. Amarel. Basic themes and problems in current AI research In: *Proc. 4th Annual AIM Workshop.* Rutgers University Press, 1978.

[2] H.N. An-Nashif, G.H. Powell. An object-oriented algorithm for automated modelling of frame structures; stiffness modelling. *Engineering and Computers*, **7**: 121–128, 1991.

[3] T. Barański, P. Sorbjan, O. Gajl, A. Radomski. Implementation of an expert system for safety evaluation of Czorsztyn and Klimkówka dams. In:, *Proc. I Conference on Safety of Hydroengineering Structures*, 263–270. Wrocław, 1993.

[4] J.H. Boose, J.M. Bradshaw. Expertise transfer and complex problems using aquinas as a knowledge acquisition workbench for knowledge-based systems. *Int. Journal on Man-Machine Studies*, **26**: 2–28, 1987.

[5] J.H. Boose, B.R. Gaines, eds. *Knowledge Acquisition Tools for Expert Systems.* Academic Press, 1988.

[6] K.W. Chau. An expert system for the design of gravity-type seawalls. *Engineering Applications of Artificial Intelligence*, **5**: 363–367, 1992.

[7] P. Coad, J. Nicola. *Object-Oriented Programming.* Prentice Hall, 1993.

[8] C.L. Dym, R.E. Levitt. Toward the integration of knowledge for engineering modelling and computation. *Engineering and Computers*, **7**: 209–224, 1991.

[9] X. Feng, T.A. Weber. Knowledge acquisition advisor (KA2): An interactive knowledge-acquisition tool for expert systems development. *Engineering Applications of Artificial Intelligence*, **6**: 507–518, 1993.

[10] G.L. Fenves. Object-oriented programming for engineering software development. *Engineering and Computers*, **6**: 1–15, 1990.

[11] B.M. Frank, T.Krauthammer. An expert system for field inspection of concrete dams. Part I: Engineering knowledge. *Engineering and Computers*, **5**: 23–39, 1989. Part II: Artificial intelligence issues. *Engineering and Computers*, **5**: 119–131, 1989.

[12] P. Friedenland. Acquisition of procedural knowledge from domain experts. In: *Proc. 7th International Joint Conference on Artificial Intelligence*, 856–851, 1981.

[13] A. Goldberg, D. Robson. *Smalltalk 80: The Language.* Addison-Wesley, 1989.

[14] I. Jacobsen et al. *Object-Oriented Software Engineering.* ACM-Press, Addison-Wesley, 1992.

[15] M. Kleiber. Computer-assisted qualitative mechanics: an exemplary simulation of a "snap-through" problem. *Computers and Structures*, 1994 (in print).

[16] M. Kleiber, Z. Kulpa. Computer-aided Qualitative Analysis: a Key to Effective Simulation and Analysis of Physical Systems? In: *Proc. Japanese-Polish Joint Seminar on Advanced Computer Simulation*, 123–130. Tokyo, 1993.

[17] Z. Kulpa, M. Sobolewski. Knowledge-directed graphical and natural language interface with a knowledge-based concurrent engineering environment. In: *Proc. CARs & FOF: 8th International Conference on CAD/CAM, Robotics and Factories of the Future*, 238–248. Metz, France, 1992.

[18] H.H. Lee, J.S. Aroda. Object-oriented programming for engineering applications. *Engineering and Computers*, **7**: 225–235, 1991.

[19] H. Ohtsubo, Y. Kawamura, A. Kubota. Development of the object-oriented FEM system — MODIFY. *Engineering and Computers*, **9**: 187–197, 1993.

[20] A. Radomski, O. Gajl, G. Pietrzak. Expert system for monitoring of Czorsztyn dam. In: *Proc. II Conference on Knowledge Engineering and Expert Systems*, 611–618. Technical University of Wrocław, Wrocław, 1993.