DISTRIBUTED KNOWLEDGE BASES AS AN INTEGRATED BUILDING SYSTEM

Christof A. Hertkorn
Rud. Otto Meyer - Systementwicklung
Institut für Industrielle Bauproduktion
Universität Karlsruhe, 7500 Karlsruhe

ABSTRACT

Changing working processes not only in manufacturing and assembly but also in office work require advanced buildings which allow a maximum of flexibility towards building structure and all its services (HVAC, telecommunications etc.). To gether with the growing importance of shared tenant services in intelligent buildings, theattempt to reach low-energy consuming buildings despite of their required flexibility in supplying a good infrastructure leads to a growing importance of "building performance". After the concentration on automated manufacturing and assembly (CAD/CAM,CIM information systems etc.) the next step is "building automation". Of course, to design, build and maintain an advanced building is too complex to be solved without computers -and even if it is done computer-based, it still requires more sophisticated programming paradigms than conventional software engineering offers. Because of the excessive complexity of the building model which has to be represented, only distributed, loosely coupled knowledge bases promise a practical apporach. This means, intelhgent buildings need to be designed, built and maintained with intelligent tools, in other words with 'Distributed Expert Systems as an Integrated Building System".

THE NEED FOR INTEGRATED BUILDING SYSTEMS

Increasing importance of shared tenant services, building and office automation as well as the growing importance of flexibility in working processes lead to the requirement of buildings that are changeable with little effort to more and more frequent changing demands towards service supplies (from telecommunications via lighting via heating to air conditioning etc.) and space (by growing or shrinking working teams, changing team and work structures, moving companies etc.). The result are decreasing life cycles of the building shell, its services, the scenery and sets. The solution are buildings which are adaptable with a minimum of effort. Together with the desired little energy consumption of a building, this leads to the claim for concentrating more on the performance of a building -especially when designing it, but as well when using -or: maintaining- the building.

Of course, advanced buildings are not easily to design, because the number of constraints is large and, even more, the number of dependencies between these constraints is exploding. For the design of such buildings, the specific characteristics of HVAC-systems, lighting, electrical and mechanical services, computer networks, telcommunications etc. as well as the estimated energy consumption based on dynamic simulations, problems like shortest paths for the building layout etc. have to be considered. Each of these systems -seen isolated of the other systems- is extremely complex for itself and difficult to oversee. The combination of these systems results in a non-manageable, complex and unstructured data network which is even too complex for computer programs.

In fact, expecially in the disciplines to create building plans by CAD systems and in enhancing building automation by DDC systems, the first steps toward integrated building systems have already started -even, if it is clear that the goal of a complete integration of several disciplines as well as of the different stages of a building -design, construction, maintenance- still seems to be far.

The concepts described here are part of the project RETEx which is partly sponsored by the Government of West Germany and carried out at the Institut für Industrielle Bauproduktion, Universität Karlsruhe, and the Company Rud. Otto Meyer, Hamburg. The project is also part of the activities at the IEA Annex 21.

MAJOR PROBLEMS FOR INTEGRATED BUILDING SYSTEMS

The main topic af an IBS is communication, in other words to exchange data between applications with completely different requirements towards data (or more general: knowledge) representation, time schedules, user and hardware interfaces. Several subsystems like CAD or DDC exist even today, but there is no direct communication between these components. In fact, there is even no vision for understanding these systems as only little parts of a large, integrated system for buildings -comparable to CIM in mechanical engineering, e.g.

Integrating varied systems covering diverging topics leads to confrontations with three problems: The combinatoric explosion of complexity by coupling and even integrating already complex systems, the need for parallel work in concurrency, and the need for distributing these parallel activities in space. These qualities have to be combined with the requirement of (relatively) easy integration of new components, where these components may be conventional programs or expert systems. An IBS has to support the extremely high dynamics of group work just like every group coordination system (compare the AMIGO MHS+ (Bogen and Weiss 88) and COSMOS (Buckley and Johnson 88) activities).

Complexity by a large amount of views

Within an IBS the specific characteristics of HVAC-systems, lighting, electrical and mechanical services, computer networks, telcommunications etc. as well as energy consumption based on dynamic simulations, problems like shortest paths for the building layout etc. have to be considered. Each of these systems -seen isolated of the other systems- is extremely complex and difficult to oversee. The combination of these systems results in a non-manageable heap which is even not solveable by computer programs. In contrast, computer applications concentrate on extremely simplified views to the real world - and still face several problems. (In artificial intelligence ABLOOS (Flemming et al. 1990) for solving layout problems in kitchens and bathrooms, e.g., or in "conventional" applications, databases manage only very restricted and pre-structured data and rarely satisfy complex structured or even new or changing requirements (Katz 1985)).

Disciplines

The most important feature to be able to "survive" in complex systems is the ability to concentrate on momentary relevant facts, solve a problem locally and afterwards decide what will be the next relevant part to concentrate on. For different problems different representations of the problem-relevant facts may be chosen. Finding out an optimal layout of rooms on a floor under consideration of the specific constraints (rooms with high/low installation requirements, natural lighting need, short paths etc.) needs other problem representations than finding out which facade element to choose whith special consideration of energy storing and daylight obtaining. Independently of the fact that all is about the same building (or even the same store or rooms), it is seen from completely different views*. Indeed, the question is, if these views on "objects" differ so extremely (like in the example above) in their prevailing context, that the information resulting of the "subjective" view within the context is more dominant than an "objective" common denominator (which consists possibly only of the name?). Actually, it is exactly the common denominator which is searched when building a database. Therefore, the approach of using a database as an intelligent commu-

*The word "view" derives form the database world. Other commonly used synonymes are: Contexts, Worlds, Aspects.

nication channel between applications and defining special views on such a database seems to be hopeless from the start, even if the database is object-oriented.

Stages

The problem of extremely diverging views for the different disciplines grows when time is added. An IBS is able to work only when the dimension "time" is supported. The three major phases of a building are: design, construction and maintenance. Each of these stages are divideable in milestones for "finding principles", "defining functions", "shape determination" and "specification" for the design phase. For each phase special views have to be established (fig. 1).

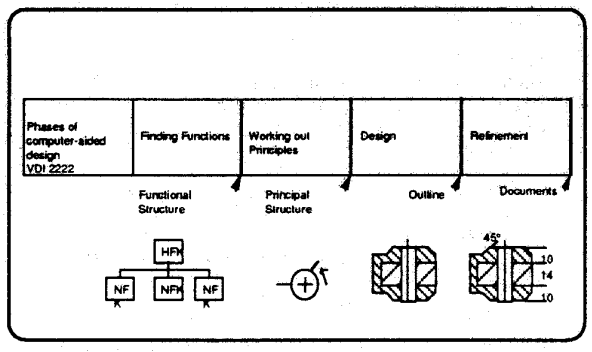

Fig. 1 - Different Design Stages (conform to VDI 2222) need different representations

Versions

The problem space covered by an IBS is even growing because very often several "would be" variants or versions of the same building (or parts of it) have to be generated to be able to decide which version (or which combination of versions) is most promising for further work. An HVAC specialist may work out three proposals for a layout of inlets and outlets for air supply, so an interior designer may discuss which solution will be promising to work on.

Concurrency

Lot's of work can be done in parallel. E.g., an HVAC specialist may work out air supply facilities for a floor at the same time when another expert works out the lighting and cabling for this or other rooms. Usually, these works won't collide but, of course, this event of collision can't be excluded. Therefore, mechanisms have to be worked out which allow the detection and handling of such conflicts. An IBS has to support concurrency.

Distribution

To support concurrency, multi-user-multi-tasking environments are indispensable. The first approximated solution might be a central "omnipotent" database server covering all aspects relevant for all views where each view forms a subset of the central database. In fact, the above described extreme complexity of the data and knowledge which has to be represented shows, that an "omnipotent" database is illusionary. Not only the complexity of represented knowledge but as well the large independence of tasks between each other shows that a common denominator, so it exists, is very little. Adding these facts leads to the demand for distributed representations of task-specific knowledge. Another aspect is the geographical spreading of the different engineering bureaus working all on the same project.

Expert Systems to support "real" design

In early design stages lots of ill-defined problems within usually largely complex problem spaces have to be solved. Usually, "thumb rules" and other representations (logic, e.g.) of experience knowledge help to come to fast approximations to good solutions. Only expert systems support the modelling of such heuristic knowledge. This can be used for diagnostic/analysis phases within the design process to find out "holes" to validate the completion of a stage, but as well for generating proposals (construction/synthesis). In both cases, the results may be incomplete and even partly incorrect, but still they help to speed up the work of a designer/engineer. Expert systems therefore act as qualified assistants while designing, and expert systems shine with their (at least theoretically) easy extendable knowledge bases by the designer himself.

Incompleteness

An IBS will never be complete (When is a building or a program completed?). Instead, the activation of the momentary most important modules will be decisive and lead to a good performance. An IBS has to be able to live with permanent incompleteness.

Unlimited Extendability

To be able to cope with permanent incompleteness permanent extendability of the system is required. It is unpredictable to foresee which new aspects/views will have to be added to the IBS.

Conclusion

An IBS has to enable users to add and remove easily modules containing dedicated problem solvers (programs, expert systems) together with a compatible both user and data interface. These modules have to work like tasks in a multi-user-multi-tasking environment allowing concurrent design with parallel processing distributed over a network of graphic workstations. Because of the diverging views/aspects the modules have to contain their own problem-specific knowledge and data representations. This leads to a physical distribution of the stored data in the spreaded modules. The comunication has to be established by a message handling system that allows the loosely coupling of modules.

PREVIOUS APPROACHES

Finding a practicable approach to an IBS leads at first to analyzing previous approaches.

Isolated Systems with dedicated Knowledge Representations

Because engineer's offices working on architecture, HVAC, sanitary, electrical etc. systems are used to work in low-budget bureaus and isolated of each other, only little and isolated computer programs have been developed. Each of these programs works with a different user interface and different data representations. To combine the input and output data between these programs, experts are required. Such experts find out which program needs which data in what form and when. The plans generated by architects, e.g., have to be interpreted for an HVAC specialist, the relevant data has to be extracted and -after certain conversion and transformation activities- fed to the different HVAC programs.

Conventional CAD supports no real Design

Especially in architecture, but as well in other engineering disciplines, CAD is used for the exact, geometrical modeling of the ready-designed object. In the domain of generating drafts, CAD works faster and more exactly than humans. In addition, nowadays it seems even to be possible to exchange geometrical data between different CAD systems without losing too much nformation. But still, no real design can be done by CAD. Except for the last step of the design process, existing computer systems are no real alternative to manual design (compare fig. 1).

A2 as a centralized integrated intelligent design system

At the ifib (institut für industrielle bauproduktion), universität karlsruhe, a prototype of a computer-aided design system was developed that includes expert system components for the design of mechanical systems as part of the building equipment (Drach 89). This prototype-program, called A2, features:

- A complex object-oriented data-model that represents the building and

the design environment with all relevant facts and knowledge about the whole design process as well as the modeling of the design object. Adaptable consistency checks and "intelligent" proposals are performed by the system.

- It supports different design levels from conceptual layout-design up to specification lists and drawings at the same time.

- A CAD-oriented user-interface is the general tool for interactive design-manipulation and control.

- Both the system-user and expert system components make use of the same set of tools to manipulate the planning world represented in the data.

Because design problems are too complex to be solved completely, exhaustive programming is impossible. The user may individually choose the level of aid by the system: it can be run as a conventional CAD-System, extended by consistency- and plausibility-checks (analysis, diagnosis) or through automatically performed design steps (synthesis, construction).

The conceptual main deficits of this system are:

- No concurrency possible. This means not only that it is impossible to parallelize algorithms with adequate effort, but as well that it is a single-user-single-tasking system.

- No distribution possible. Because all data and knowledge is stored as one semantical network on one machine, each separation of data means extreme effort.

- The single-tasking system leads to a synchronous check of constraints -with two consequences: with each new constraint exploding response times, very often checking of constraints which are temporarily not important.

- No Extension possible. The frontier of the capabilities of the used expert system shell is already reached -far beyond a serious application is possible.

The experiences made with the A2 prototype lead to the considerations presented in this article.

APPROACHES IN RESEARCH

After having determined that previous approaches have not reached the requirements of a practicable IBS, a closer look to actually discussed ideas follows.

Asynchronous message passing (send) replaces synchronous message passing (call)

Present object-oriented programming environments support only the conventional object-oriented programming manner (Booch 90). "Conventionally object-oriented" means that objects that might inherit attributes from ancestors, hold methods which may be activated by calls from the outside and even cover demons which "fire" when specifc events like the alteration of a value happen. Especially, such demons may call methods in other objects and so perform automatically side-effects after changes of an object are made. The problem of so organized passive objects, which act only when an owned method is invoked, is that all other objects have to wait until the method (which may even call other methods for those it waits) is ready (single-thread-of-control). Especially the caller of the method (or the invoker or the fired demons) has to wait until the called method or side-effects are performed - even if these side-effects are not interesting at that moment.

Whereas it is sometimes useful to wait until side-effects are performed (which means that side-effects are used as subroutines) it very often is superflous to wait for momentary non-crucial results. The solution of this problem is the elevation of synchronous message passing of type call to the asynchronous "true" message passing of type send. This, of course, leads to a multiple-thread-of-control that results in a process-oriented paradigm with all consequences on programming covering aspects like avoiding or detecting and removing dead-locks, assigning priorities, handling asynchronous interrupts and managing message queues (Kemper et al. 90).

Passive Objects become active Processes

To allow asynchronous message passing demands that passive objects turn to active objects which hold a mailbox for receiving messages and a queue for sending messages. Active objects therefore mean nothing else than independent processes or tasks that do their job without explicit induction from the outside, so-called "autonomous objects" (Kemper et al. 90) or "actors" (Agha 86).

Systems that allow "true" message passing are not available yet. Systems allowing message sending even through networks as well as between objects of different tasks are not in sight, too. In fact, research projects facing these topics will last for several years and are seen scepticly by "practicians".

The conventional research still dreams of a common denominator of all data that has something to do with the whole product from the design to the manufacturing, assembly, delivery and maintenance stage(STEP/EXPRESS, comp. (Schenck 88)). The same approach is tried for buildings (Richter 89) and leads to directions and rules that can't be observed in practice.

PRAGMATIC APPROACH: DISTRIBUTED KNOWLEDGE BASES AS AN INTEGRATED BUILDING SYSTEM

As described above, implementations of an IBS can't be based on previous arised computer-based tools and need asynchronous, loosely coupled, distributable active objects. But, shells supporting such programming paradigms don't exist today. Therefore, the concepts presented here are based on existing tools but extended by self-made capabilities especially featuring network-transparent message-passing and object storage.

The central structure is called a module. A module is an application which works as a conventionally programmed process and solves a dedicated class of problems. In other words, a module of the IBS is nothing else than an usual program. The difference is only, that all modules use the same toolkit for the user interface and the same interface to object storage. These features of modules lead to significant advantages over other programs: The user has the same "look and feel" on all applications and therefore much less effort for learning a new or not well known program. In addition, the communication interfaces between modules are syntactically standardized. Because of the mutual independence of modules it is possible to omit momentary unimportant modules and to add them again when they are needed.

The system architecture is shown in fig. 2.

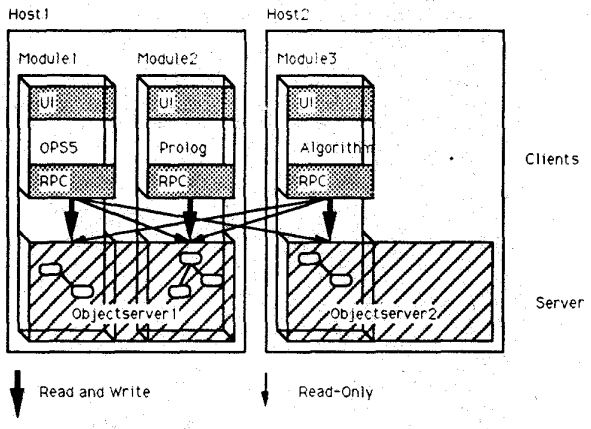

Fig. 2 - Modular System Architecture

Modules

Modules work as agents which have special know-how for solving specific problems (Connecting a sink to a waste-water pipe, e.g.). Each module has got a user interface cut to that domain that is covered by the module. In addition, each module may store data and "knowledge" in form of objects in

an own server to publish its results and required own data which might interest other modules. This means that modules consist of three major parts: The user-interface, the program (which may be even a rule system, e.g.) and an object store.

Domain-specific knowledge

Because of the highly varying problem classes covered by an IBS very different knowledge representations are required. Whereas algorithms implemented in conventional programs solve lots of problems, especially in the design stage of the building or its components knowledge representations like forward- or backward-chaining rules, logic or semantic networks are required for a quicker orientation in complexely structured problem spaces. The structure and methods of programming of modules therefore can't be prescribed or ruled and, in contrast, has to be hold open. The only instruction for programming a module is to use the user interface toolkit as well as storing data of public interest using the unique interface of storing and retrieving data represented as uniquely structured objects.

Compatible user interfaces

To reach the same "look and feel" for the user on all modules, the same object-oriented toolkit is used for all modules. For the currently implemented prototype, a self-developed extension of the Interviews-Toolkit (InterViews89) is chosen. InterViews is public domain, delivered in source code and based on the X11 standard. The main constructs of the Toolkit are the class definitions interactor and scene. Both define a protocol for combining interactive behaviors. The extension covers the integration of C++-objects in the lisp and knowledge craft environment.

Unique, but distributed data storage for all modules

The same interface between programs and data storage for all modules formulates a platform which may lead to a maximum of communication dense between modules if required. To allow simple but powerful and easily extendable data access, a set of orthogonal commands is realized. The basic commands of managing data are "get", "add", "delete" together with the predicate "is". These primitive operations work on the whole object structure that consists of the object-name itself, a list of attributes (that are called slots) and correlated to each slot a list of values (fig. 3). These values again are not restricted and may even be object-names again (which elevates the attributes of an object to relations between objects). Whereas object-names and values are strings, the typing of the values is not at all restricted (like in Lisp). Values may be names or numbers or lists consisting of other values. Especially they might even contain source-code (preferrable Lisp) which might be fetched and evaluated or compiled when needed.

The communication between modules has to be handled by channels. Modules consist therefore of a client (containing the module's program and the user interface) and on the other hand an own server -serving as a communication channel- where the objects are stored.

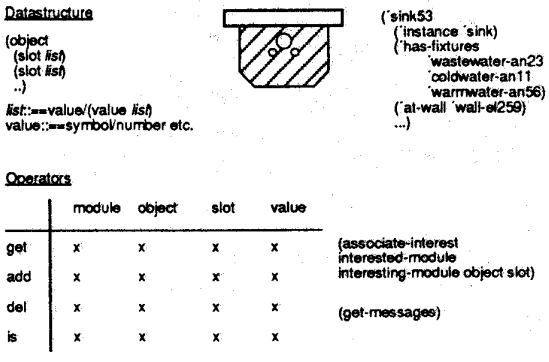

Fig. 3 - Data Structure and Orthogonal Operations

The communication for the client-server-model is realized by Remote Procedure Calls (Sun 90). A client-server model represents a good solution separating the problem-solving part in the client and the used and to be published data in an own object server. Still, no centralized server is in use but each module keeps its own server containing all facts of public interest but dedicated to the problem domain of that module (fig. 2).

Communication

Of course, an IBS doesn't already work when there are lots of compatible modules using the same user and data interface and keeping their publishable data in private servers. Therefore, the elementary operations for managing objects are able to work also on objects of other modules, too. To solve problems like mutual exclusion, two approaches are concerned: No write-access to objects of other modules (which implies no write-access to their slots and values, too), or the establishing of a transaction management. Because of the high effort realizing transaction concepts and the need for a central transaction server (Engelien 91) the first idea is preferred. This idea is conform to the encapsulation paradigm in object-oriented programming (Booch 91), because every agent has got the right to change things only within his own scope. To improve the communication with regard to unexpected but important events, so-called interests provide that automatic messages arrive when the alteration of values in a specific combination of module, object and slot is carried out in that module.

Ubiquity of modules

To fetch information covered by other modules it is assumed that each module is able to find out everything stored in the public servers of each of the modules. Therefore, whenever a module is added it's name is introduced to all other existing modules. By get-operations each module may find out the objects published by the new module as well as their actual state and hence may draw conclusions from that.

Interests for interrupts

Generally, two communication techniques are possible: Each communication partner (here: module) asks in a cyclic manner all other modules if things have changed ("polling"), or each communication partner enforces interrupts within another module to deliver a new message which is assumed to be interesting for this module. The IBS described here will allow both. Whereas polling is carried out be cyclic "get"-operations, interrupts managing asynchronous events are also possible.

As a first step, a module has to find out on which combination of module, object, slot it is interested. E.g., a module handling the connecting of waste-water pipes may be interested in the position of sinks. Of course, a cyclical fetching of the position of all sinks would consume to much time and effort because it may be assumed that sinks won't change their position to often when the connection stage of the piping has already begun. Therefore, it is easier to install an interest at the position slot of the sink object used in the module "bathroom layout": (associate-interest 'waste-water-connection 'bathrom-layout 'sink 'position). This causes the bathroom-layout module to install an "if-added"-demon at that slot. This demon fires always when a change of a value in that slot is carried out and sends a message to the specified waste-water connection module.

Summary

Modules represent a pragmatic compromise between existing tools (programs and expert systems embedded in standardized user interface toolkits and common client-server handling libraries (Sun 90)) and requirements of asynchronous communication between autonomous objects recognized by actual research activities. Each module serves as an active specialist, works on specific problems and is able to exchange anyhow structured data. The structure is based on a general lisp-like object syntax (often known as schema) that allows any extension possible (see fig. 3).

Another great advantage is the possibility to react on momentary interesting problems by letting less interesting modules out or adding special modules when required -at any time. The hopeless attempt to model all possibly interesting aspects in one centralized database is replaced by the pragmatic combination of momentarily required specialists, hence modules. This has the wished side-effect of distribution of the used facts, increasing performance by parallelisation of processes/modules and the establishing of a multi-user-environment working on different places at the same time.

EXAMPLE

The way how communication and distributed knowledge-based problem solving might work is shown in pict. 4. Imagine the time axis vertically. Horizontally, graphical representation of the actual state of three modules (here "bathroom layout", "waste water plumbing layout", "cold/warm water plumbing layout") at the same time is shown.

Fig. 4a shows a ready-planned situation of the plumbing underneath a bathroom. It is assumed that the bathroom layout designer (who might be a rule in an expert system, e.g.) decides to move the sink to the other wall. The consequence is seen in fig. 4b. It is assumed that the two right-hand modules have associated interests to the slot position of the object sink within the bathroom design module. Therefore, with the move of the sink the two other modules react immediately by displaying their private graphical representation of exactly that sink at the new location. In fig. 4c1 the piping modules have not only detected the planning deficit of unconnected pipes but already -and independently of each other- generated new connection lines in parallel. As may be seen, these changes have not reached the other modules -obviously because no interests were associated. But, happily each module polls cyclically relevant values of interesting objects and so an upgrade can be carried out like in fig. 4d1.

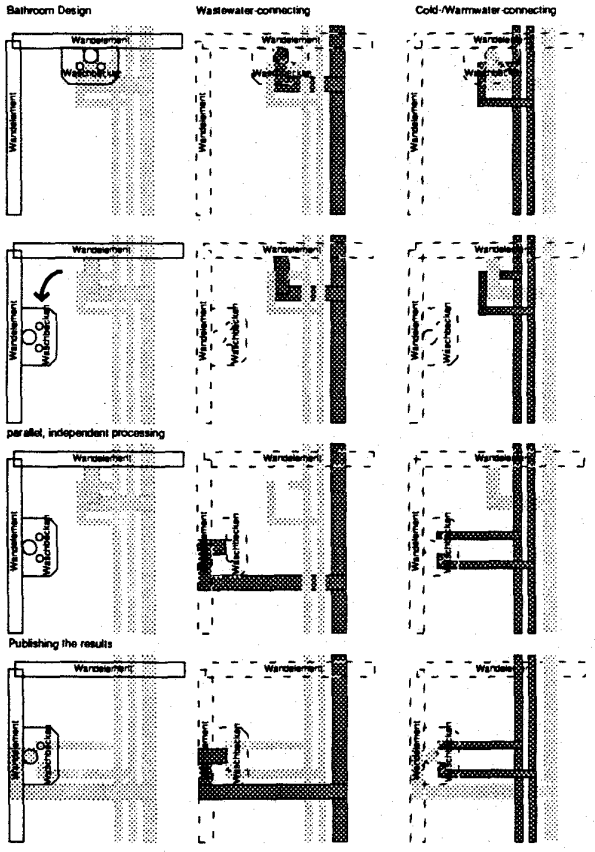

Fig. 4a,b,c1,d1 - SuccessfulRedesign

Of course, such operations do not work together so happily all the time. So, let's assume that instead of the ation carried out in the piping modules at fig. 4c1 they generated connections between sink and the main pipe as shown in fig. 4c2. Because of the lack of information, that may occur when no interests are anchored. But, as seen in fig. 4d2 and 4e, after the next polling cycle the conflict will be detected and cleared up -assuming that competences between (big) waste water pipes and (little) cold/warm water pipes are clarified in advance and, of course, programmed...

SUMMARY

Whereas existing software components and development tools don't suffice fundamental requirements of an IBS and other research approaches are not matured enough, the concept proposed in this paper represents a pragmatic approach to a computer-based building design support system.

Especially to centralize all data in a single database is declined. Instead a decentralized way of storing data there where it is mostly needed without any central institutions is preferred. Together with a general object structure this leads to an extremely flexible structure, adaptable to momentary occuring problems -ideal for the extremely high dynamic of group work typical for building design.

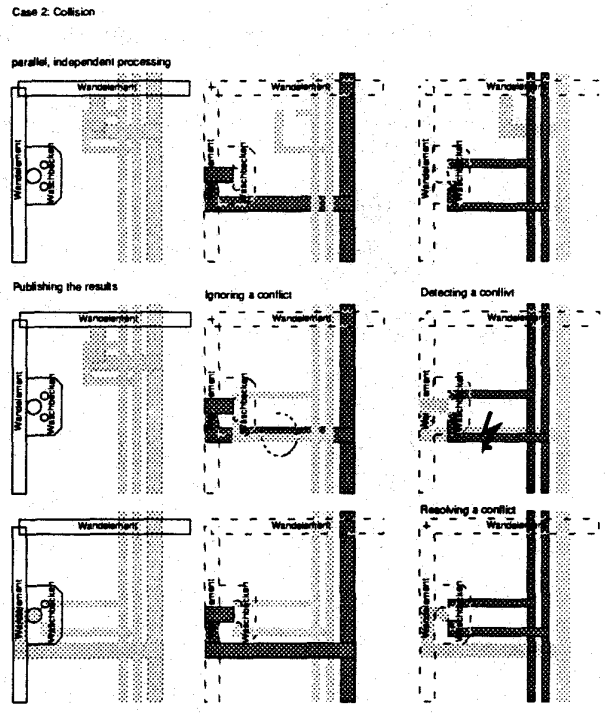

Fig. 4 c2,d2,e Detecting and Solving Conflicts

REFERENCES

Agha, G.A., 1986. *ACTORS: AModel of Concurrent Computation in Distributed Systems*. The MIT Press, Cambridge, Ma.

Bogen, M. 1988. "Group Coordination in a Distributed Environment." In *Research into Networks and Distributed Applications - EUTECO '88*, R. Speth, ed. North-Holland, Amsterdam, 185-193

Booch, G. 1991. *Object-Oriented Design with Applications*. Benjamin/ Cummings Publishing, Redwood City.

Buckley, P. and P. Johnson. 1988. "Analysis of Communications Tasks in the COSMOS Project." In *Research into Networks and Distributed Applications - EUTECO '88*, R. Speth, ed. North-Holland, Amsterdam, 185-193

Drach, A. and L. Hovestadt. 1989. "Intelligente CAD-Systeme-Instrumente für die Planung und Verwaltung komplexer Gebäude". InProceedings of *Intelligent Building Symposium 1989* (Karlsruhe, Germany, Oct. 12./13.. Kraus, Karlsruhe, FRG, Ch. 9.

Engelien, D. 1991. "Entwurf und exemplarische Implementierung der Interprozeßkommunikation für ein verteiltes Gebäudeentwurfssystem." Research Report of institut für industrielle bauproduktion, universität karlsruhe, Karlsruhe, FRG.

Flemming, U. et al. 1989. "A Generative Expert System for the Design of Building Layouts (Final Report)." Technical Report EDRC 48-15-89, Engineering Design Research Center, Carnegie Mellon University, Pittsburgh, PA, USA 1989.

Linton, Markt A. et al. 1989. "Composing User Interfaces with InterViews." *IEEE Computer*, February 1989, p. 8-22.

Katz, R.H. 1985. *Information Management in Engineering Design*. Springer-Verlag, Berlin, FRG.

Kemper, A. et al. 1990. "Autonomy over Ubiquity: Coping with the Complexity of a Distributed World". In *Proceedings of the International Conference on Entity-Relationship Approach* (Lausanne, Switzerland, Oct.).

Richter, P. 1989. "Integrierte Informationsstruktur im Bauwesen." Research Report. Projekt ISYBAU, Mettmann, FRG.

Schenck, D. 1988. "Information Modelling Language EXPRESS". Technical Report ISO TC 184/SC4/WG1 Document No. 287 (Oct.).

Sun microsystems, March 1990. "Network Programming Guide". Part Number 800-3850-10. Sun microsystems, Inc.

VDI (Verein Deutscher Ingenieure). *Vorgehensplan für die Produktentwicklung*.VDI-Richtlinie 2222.