KNOWLEDGE–BASED CODE CHECKING PROGRAM FOR BUILDING DESIGN

Wei Dai and Stephen Oakes
CSIRO Building, Construction and Engineering
PO Box 56, Highett, Vic. 3190, Australia
Email: {wei.dai, stephen.oakes}@dbce.csiro.au

INTRODUCTION

The purpose of a building design process is to create a description that satisfies building functional specifications, requirements on performance and resource usage, as well as restrictions such as building regulations. The need to develop intelligent computer programs to check compliance of a design with building codes was specified by Niwa (1987), Sharpe and Oakes (1995). A previously proposed solution was to convert building codes into 'hard-coded' programs which would significantly restrict the scope of interactions with related building construction programs.

There is a growing need within construction industries to overcome communication barriers between different disciplines and eventually create an integrated design environment. This paper addresses part of the concern, i.e. to integrate building code checking into a design environment. The work is based on a commercial software product called BCAider®, whose major functionality is to help check the validity of a building design, based upon the clauses in the Building Code of Australia (BCA). The input for the program comes entirely from its user. Upon its commercial release, BCAider gained successful market penetration at an early stage. The major users are local governments in different states of Australia. However, it faces difficulties in broadening its market scope. This is mainly due to the limitations such as inaccessibility by other products and rigid (most of the time static) problem-solving structures. To overcome those limitations, much more intelligence and dynamic problem-solving capabilities are required.

The work described in this paper is about the future generation of BCAider, referred to here as 'BCAider Design'. BCAider Design employs state-of-the-art technologies, from artificial intelligence, object database and principled software design and implementation. It offers a solution to the integration of code checking into a user's work environment and responds intelligently to various user tasks. BCAider Design is equipped with a more advanced knowledge base capable of understanding building regulations at a very fine level, and processing design details. It has a domain task description component which converts external tasks (such as those from CAD environments) into its internal format. The inference engine, upon receiving task descriptions, deposes the design data into various 'regional designs' according to building regulations stored in the knowledge base. The prototype system demonstrates some of the desired features and is moving towards commercialisation.

RESEARCH INITIATIVES

The main objective of BCAider Design is to add the much needed intelligence (from a user's point of view) to the building code checking program and integrate this functionality with external software packages (i.e. CAD systems) within a design process. The addition of intelligence to BCAider (the current work of BCAider Design) can only be accomplished if it has access to the plans of the building under consideration. This access must allow BCAider Design to generate queries (e.g. 'Is there a door between room_1 and room_2?') and to store its own information, such as the distance from the furthermost corner of a room to the nearest 'required exit'. There must also be a mapping between the terminology used within BCAider Design (and implicitly the BCA) and the information structures used within the database that BCAider Design is accessing.

BACKGROUND INFORMATION

BCAider Design is being developed using advanced technologies from knowledge-based systems which include knowledge representation, domain task representation, system state representation and inference control. Each component contributes one aspect of the knowledge-based system's problem-solving process.

Rule-based representation

Rule-based representation is used to describe building code semantics. The representation template is given in Table 1.

Table 1: Rule representation template

RuleID
{ LHS
Premise1: OAV;
Premise2: OAV;
...
$Premise_m$: OAV;
RHS
Conclusion1: OAV;
Conclusion2: OAV;
...
$Conclusion_n$: OAV;
ACT
Action1: string;
Action2: string;
...
$Action_p$: string
}

A rule contains several parts. The LHS (left-hand side) describes all the conditions that a rule must satisfy before conclusions can be made. The RHS (right-hand side) contains all the conclusions. Each LHS or RHS unit (e.g. a premise or a conclusion) is described

by a type of object-attribute value (OAV). The other part of the rule is the action (ACT) section. The ACT contains all the actions to be performed once the rule conditions are satisfactory. Each action has a simple type of string.

Domain task representation

Domain task representation is used to describe a building design. The representation template is given in Table 2.

Table 2: Domain task representation template

ObjectID
{
Property1: AV;
Property2: AV;
...
$Property_k$: AV;
}

The domain task to be processed by the knowledge-based system is described in terms of data objects. A data object is described by an arbitrary number of the properties (see Table 2). Each property has a type of attribute value (AV). The power of expressiveness of attribute-value pairs was discussed, from a design perspective, in the FBS (functional, behavioral and structural attributes) model (Gero 1990).

System state

This describes the conclusions reached by the system at a particular point of time. This information is in terms of system state data objects, each of which is defined by an OAV.

Inference control

Inference control concerns the knowledge-based system's problem-solving process. The program performing the inference task is called an inference engine. Upon receiving a domain task, the inference engine applies relevant knowledge to attempt a solution. The process of applying knowledge may require additional information which cannot be generated through deductions based on the knowledge the inference engine has. In this case, system state information (general facts) and domain task information (e.g. a building design) may help answer some of the system's questions. The last attempt (if the required information still cannot be obtained) is to obtain information directly from the user. Difference problem-solving strategies can be used in the inference engine, e.g. goal-directed and event-driven inference. Goal-directed inference uses backward-chaining to solve a problem, i.e. the task (goal) initiates the solution plans and results. Event-driven inference attempts the solutions in a forward-chaining fashion where any outcome is possible, such as a random number of design parts contradictory to building regulations.

State-of-the-art computing technology

BCAder Design is based on the research achievements in artificial intelligence, software reuse, software development environment, object database, etc. The outcome of these efforts is that high quality software products are produced in a cost-effective way.

The current state of the system's internal implementation includes domain-dependent and domain-independent graphic user interfaces linked with knowledge-based and database facilities. Software tools are also available for entering and editing knowledge and design data, and performing inference tasks based on event-driven or goal-directed strategies, or a combination of both.

In the application section, we shall see examples of combining the above-mentioned knowledge-based system components to solve building code compliance problems.

KNOWLEDGE SOURCE: BUILDING CODE OF AUSTRALIA

The Building Code of Australia (BCA) consists of eight sections. Each section contains a number of parts, and each part contains a number of clauses. A small number of clauses refer to a corresponding 'specification'. Specifications each also contain a number of clauses. The structure of the BCA is shown in Figure 1. Overall, there are about 300 clauses and 17 specifications in the BCA. In addition, each of the Australian states varies the national version of the BCA to some extent, generally by adding new clauses, or by varying existing ones.

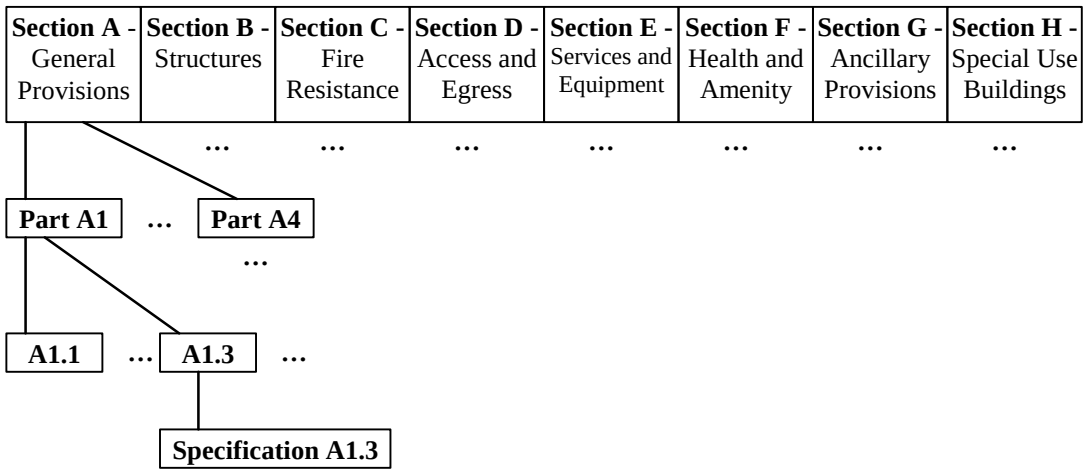

Figure 1: Structure of the BCA.

BCAider operates at the clause level, allowing a user to select a clause and then 'process' it. (BCAider Design is not restricted to operating at the clause level. It may operate at the sub-clause level, depending upon the clause complexity and applicability.) Processing involves asking the user a series of questions in order to come to a conclusion. Questions usually require the user to select from a set of 'Yes/No'

buttons, but occasionally require the user to enter other forms of information, such as typing in a number. After the user enters enough information, the result of the compliance check ('Complied' or 'Not Complied') is displayed.

At any stage during the processing of a clause, the user may read the contents of the clause by activating the electronic version of the BCA, which is built into BCAider. Hypertext references to definitions, Australian Standards and other clauses are also available.

RELATED WORK: AUTOAIDER

The current BCAider product on the market incorporates a limited interface to the popular CAD program AutoCAD. Known as AutoAider, this interface allows the AutoCAD user to identify BCA clauses that need to be checked as they work on a design. Although the interface is not particularly sophisticated, it is useful for a percentage of BCAider users.

AutoAider requires two items to be set up before it can be used. First, a small module has to be installed in AutoCAD. This module monitors any changes made to the design, and activates when the user inserts a new object that AutoAider knows about. Secondly, the AutoCAD user must use a set of predefined blocks when inserting objects (such as doors or windows) into their CAD design. Each such block has a type associated with it, and when it is inserted, AutoAider looks up a list of clauses that are associated with it. That list of applicable clauses is then displayed in a separate window for AutoAider. Once a list of clauses is displayed by AutoAider, the user may select one and then start up BCAider to process it, or jump directly to the electronic version of the BCA to read the clause.

AutoAider does not attempt to directly interpret an AutoCAD design; nor can it work on pre-existing designs. Indeed, any attempt to interpret a CAD design must rely on the use of at least a minimal form of object-orientation by the CAD application. AutoAider does little more than attempt to alert the user to a set of clauses that need to be checked when a particular type of object is inserted into a building design.

BCAIDER DESIGN FUNCTIONALITY SPECIFICATION

There were several technology breakthroughs with the BCAider Design system, i.e. building design representation, building code representation and event-driven inference. Major system functionality is described as follows.

Upon receiving a design checking task from a user, the inference engine decomposes the design information into smaller parts (called regional designs) that attract immediate BCA solutions. The result is that any part of the design which is not complying with the BCA is identified, with the design content and related BCA clauses displayed. The new BCA knowledge base stores most of the intelligence required, and is capable of describing building regulation semantics in fine detail to allow the system to interact with external systems (such as CAD) effectively. It can be used for a diverse range of applications and is easily maintainable.

All the system data (e.g. knowledge, design data, inference data) are stored and retrieved through a commercial object database system offering data persistency throughout any stages of computing. The system is thus capable of handling large-scale, high-performance application requirements, and uses computing resources very efficiently.

SYSTEM USABILITY

An important feature of BCAider Design is to provide interactive support to users by tracking the progress of a design. When aspects of the design are recognised by BCAider Design (using long-established pattern-matching techniques from AI research), it can then initiate a check on the legality of the particular component or element. The reporting of conflicts with the BCA is interchangeable between a work environment (i.e. CAD) and testing window where compliance messages are displayed, since in many circumstances designers knowingly breach BCA regulations while developing a design as particular problems are studied. BCAider Design still needs to maintain a current model of the design so that there are not lengthy delays in producing reports when requested.

It is currently envisaged that BCAider Design will run as a separate process to CAD systems, using streams to communicate between the processes. The use of streams will provide a great deal of flexibility; it may be possible to use the hardware/operating-system independence emerging under the Java language to provide client/server-type support to a wider range of systems than can currently be supported.

The initial structure of the communication mechanism will use bi-directional streams. The CAD systems will relay the commands issued by the user to BCAider Design. When BCAider Design recognises an action that is relevant, such as the creation of a new door, BCAider Design will initiate checking on that object. BCAider Design will then monitor the state of the relevant objects to check whether they comply with the BCA. When a report is requested by the user, BCAider Design can then generate this through the CAD system or as a text file on the hard disk describing the objects checked and whether they comply with the BCA or not.

COMMUNICATION COMPATIBILIY WITH EXTERNAL SYSTEMS

An important strength of BCAider Design is that it has been designed to work within a design environment. This is because of the intelligence incorporated in the system that is capable of understanding a building design at the design level of BCA. The intelligence comes from the descriptive power of a new BCA knowledge base.

The design task representation module of BCAider Design receives the ‘converted’ building design information originated from a CAD system. The information received is stored in terms of data objects and processed by the inference engine. The conversion process aims at translating CAD output into BCAider Design internal format which is currently designed to be compatible with international standard STEP (Standard for the Exchange of Product Model Data, ISO 10303:11, ISO 10303:21).

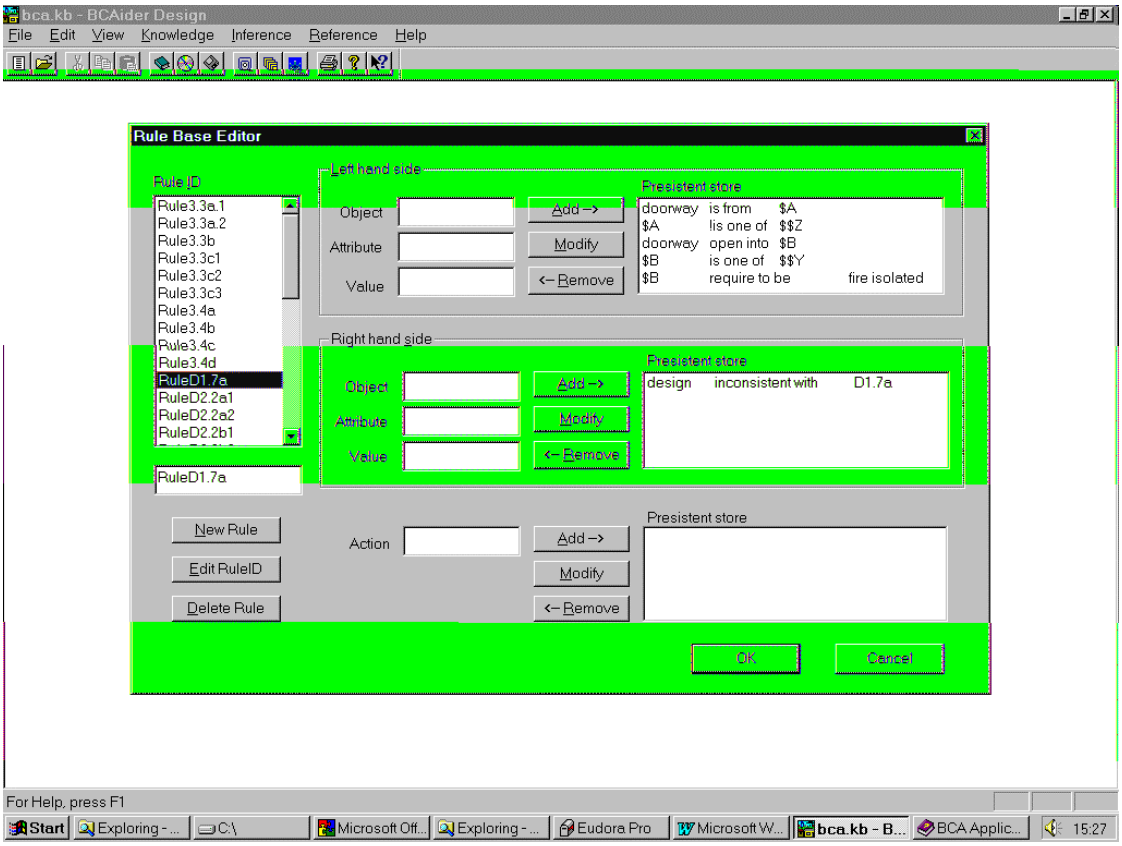

Figure 2: Content of Rule D1.7(a).

IAI (International Alliance for Interoperability) is developing the Industry Foundation Class (IFC) for the building construction industry. The IFC is based on the same technology (e.g. Express) as the STEP standards. IAI is focused solely on the building construction industry. This reduces the scope of the projects and removes much of the complexity that the STEP projects must deal with.

The development of the BCAider Design two-way communication mechanism is in line with international progress in STEP and IFC. Inference results can then be sent via the same translation/conversion mechanism to a CAD environment. Work on communication between BCAider Design and CAD systems is in progress, and will be a major focus of research in the future.

RUN-TIME EXAMPLE

In this section, we use BCA Clause D1.7(a) as an example to show how a practical problem is solved by BCAider Design. The BCA clause is given below.

> Clause D1.7
>
> (a) A doorway from a room must not open directly into a stairway, passageway or ramp that is required to be fire-isolated unless it is from-

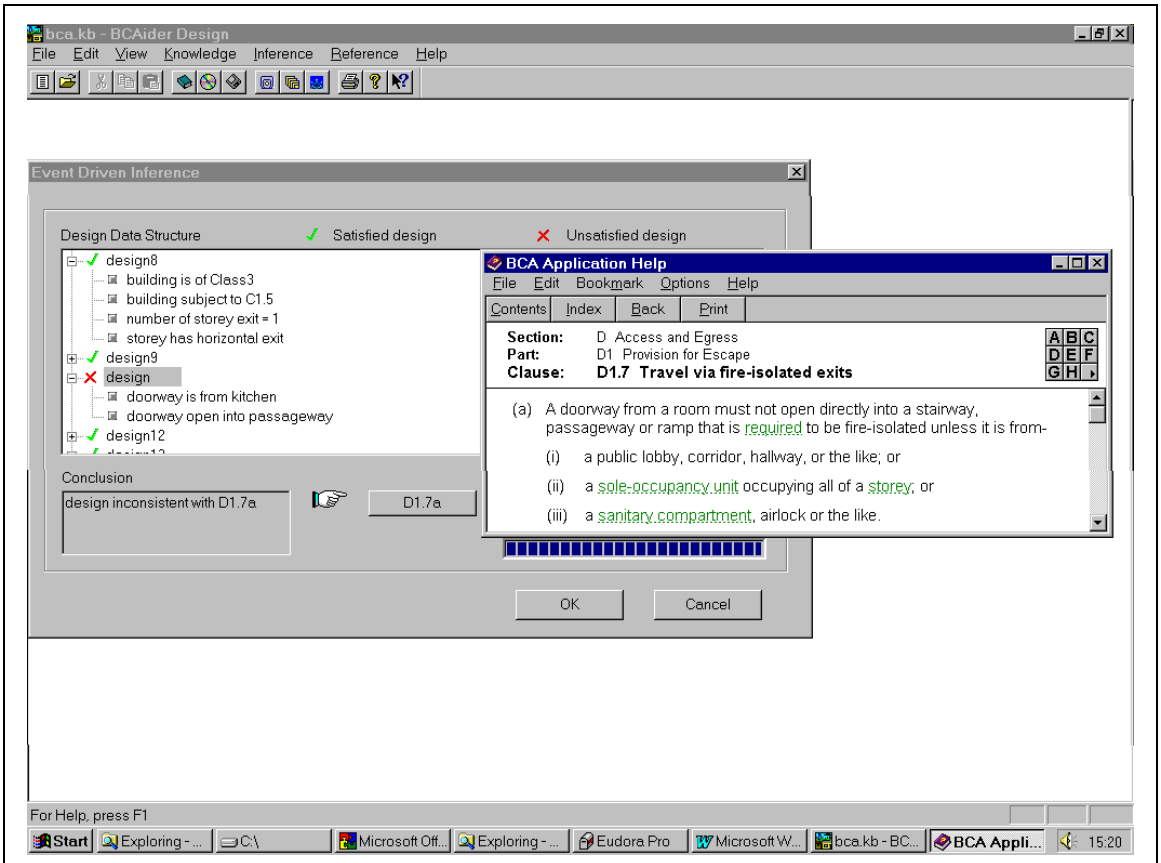

Figure 3: BCAider design run-time example.

(i) a public lobby, corridor, hallway, or the like; or

(ii) a sole-occupancy unit occupying all of a storey; or

(iii) a sanitary compartment, airlock or the like.

The relevant computable BCA rule for the above clause is presented in **Error! Reference source not found.**. In the example rule, three special symbols are used: '$$' for static values binding, '$' for dynamic value binding, and '!' is a logical NOT.

The inference results, i.e. the detailed checking of the received design information, are given in **Error! Reference source not found.**.

CONCLUSION

The paper describes the use of state-of-the-art information technologies used for building construction applications. These technologies were used in the development of the BCAider Design system, whose main strengths are its capability to understand a building design at BCA (detailed semantics) design level, and its potential to support communications with external application systems such as CAD systems.

ACKNOWLEDGMENTS

The authors would like to thank Robin Drogemuller of James Cook University for earlier discussion and comments leading to the paper. Thanks are also due to many CSIRO colleagues who have supported this work, in particular Kevin Gu, John Mashford, Michael Ambrose and Kwok Keung Yum.

REFERENCES

Gero, J. (1990). Design prototypes: a knowledge representation schema for design. *AI Magazine*, **11**(4), 26–36.

Niwa, K. (1987). Use of artificial intelligence confirming compliance to building codes during process of making architectural drawings. *Proc. Int. Conf. and Exhibition of Artificial Intelligence*, Tokyo, Japan, pp. 160–162.

Sharpe, R., and Oakes, S. (1995). Advanced IT processing of Australian standards and regulations, *International Journal of Construction Information Technology*, **3**(1), 73–89.