2006

Automated code checking and accessibility

Robin Drogemuller

Lan Ding
University of Wollongong, lding@uow.edu.au

Chapter 8
Automated code checking and accessibility

Robin Drogemuller and Lan Ding

Introduction

Ever since the introduction of architectural CAD (Computer Aided Design) systems, building designers have dreamt that the onerous task of checking building designs for compliance with the relevant building codes and standards could be automated. The reasons for this are obvious – the Building Code of Australia (BCA), as an example, contains over 4,000 clauses filling several thousand pages of text. These clauses reference over 1,000 separate standards, with options available in the standards that need to be explicitly selected within the contract documentation. Failure to comply with provisions in the building code or the referenced standards can mean expensive rework on site or even after completion of the project. This may have significant financial impacts on all parties involved in the building project.

The ability to automate code checking would also reduce problems with human interpretation of codes, where different people understand the same text in different ways. Reading the history of modern architecture leads to the conclusion that innovative and unusual buildings are often given a more rigorous analysis than more 'normal' buildings. Automating code checking could thus reduce the likelihood of errors in projects and improve the interpretation and application of codes.

The interest in automated code checking has led to a relatively long history of research in this area. Work at the NBC in Canada (National Building Codes) and at NIST in the United States (National Institute of Standards and Technology) started in the 1980s, as did research at CSIRO in Australia. There have been a number of implementations of knowledge-based systems arising from these research projects, some of which, such as BCAider, were commercialised and used in live projects. This first generation of implementations required the user to enter the characteristics of the project since they did not support automatic extraction of information from CAD data. The major reason for this was the lack of semantics (computer interpretable content) in the CAD systems available at that time. The introduction of object-oriented CAD systems since the 1990s has changed this situation where it is now feasible to build automated code checking systems. Three of these are described in this chapter.

Building codes and standards

Internationally, building codes and standards can be divided into two groups, prescriptive and performance-based codes. A prescriptive code states the requirements explicitly. A designer must comply with the provisions as given. Checking against a prescriptive code is relatively easy. A design either complies with one of the provisions explicitly stated within the code or it does not. In the latter case, the design will need to be modified. The rapid change in the available technologies within the building industry has lead to a realisation that prescriptive codes potentially restrict innovation and the uptake of new technologies. The problem with prescriptive codes is captured in the title 'Does an Ice Hotel Require a Sprinkler System?' (Rauch, 2005). The limitations of prescriptive codes have led to a movement internationally to performance-based codes.

While there is no one structure that matches all performance-based codes, the structure of the Building Code of Australia provides a useful example (Figure 8.1). At the top level are the objectives, specifying the goals of the code, then the functional statements, setting out how this should be achieved and then the performance requirements, stating how the functional requirements will be measured. Typically, the number of provisions increases as the hierarchy deepens. All of these are specified in qualitative terms. Since qualitative descriptions are open to interpretation, and hence dispute between designers and certifiers, deemed-to-satisfy solutions are also provided that give quantitative requirements to meet the requirements of the code. Where a designer wants to use a different method than that allowed in the deemed-to-satisfy provisions, the designer may propose an alternative solution. This is the major difference between performance-based codes and prescriptive codes. In a prescriptive code there is little or no flexibility in what may or may not be constructed.

Alternative Solutions, under the BCA can be certified in various ways (Figure 8.2). Comparable systems exist in other performance-based regulations. Figure 8.3 gives extracts from the BCA showing the hierarchy of requirements for disabled access from objectives down to deemed-to-satisfy. In this case, AS1428.1 is accepted as one possible solution to the stated objective. Of course, other solutions that meet equivalent requirements are equally acceptable.

Figure 8.1 Formal structure of the Building Code of Australia.

Figure 8.2 Alternative solutions under the BCA.

The type of building code has a significant effect on how an automated code checking system is implemented. Automated checking is simpler under a prescriptive system since the requirements are explicit and the design either meets these requirements or it does not. Under a performance-based system, there can never be a definitive answer since it is only possible to check against deemed-to-satisfy provisions and a designer may be proposing an alternative solution. This means that there must be a higher degree of user interaction in a system that checks against a performance-based code so that the designer can identify which methods of compliance have been chosen. In a fully automated system, meta-data would need to be stored to indicate which parts of a code or standard are met by a particular deemed-to-satisfy.

User requirements

There are three groups of people whose needs must be recognised when developing a code checking system. Designers will use code checking systems to test whether a design meets the requirements. Ideally, the system will also assist in checking partial designs to avoid backtracking when a detailed design is found not to comply.

Building compliance checkers need to validate completed designs when the design is submitted for building approval. In jurisdictions where the building compliance checking system has been de-regulated, such as some states of

Objective:

DO1

The Objective of this Section is to –

(a) provide, as far as is reasonable, people with safe, equitable and dignified access to –

(i) a building; and

(ii) the services and facilities within a building; and

(b) …

Functional statement:

DF1

A building is to provide, as far as is reasonable –

(a) safe; and

(b) equitable and dignified, access for people to the services and facilities within.

Performance requirements:

…

DP7

Accessways must be provided, as far as is reasonable, to and within a building which –

(a) have features to enable people with disabilities to safely, equitably and with dignity –

(i) approach the building from the road boundary and from any carparking spaces associated with the building; and

(ii) access work and public spaces, accommodation and facilities for personal hygiene; and

(b) …

Deemed-to-satisfy provisions:

D3.2

General building access requirements

(a) Buildings must be accessible as required Table D3.2.

(b) Parts of buildings required to be accessible must comply with this Part and AS 1428.1.

(c) …

Figure 8.3 Hierarchy of requirements within BCA.

Australia, building compliance checkers often provide advice to design teams, in which case their needs are identical to designers.

During development of Design Check (described later), both designers and compliance checkers indicated that they wanted to be able to analyse a design in various ways, through identifying which objects did or did not comply with specific clause(s) and also by selecting types of objects and checking which clauses complied/did not comply. Both groups also preferred that the entire code should be stored and all requirements met in one operation. They did not want to remember that some clauses were covered by an automated system while they would have to manually remember to check other clauses.

Compliance checkers will often audit designs more thoroughly than designers. Compliance checkers indicated that they also wanted to be able to examine individual objects or types of objects and to check which rules had or had not been run against the objects.

The final group of users is the people writing the rules which encode the provisions of codes and standards, whether these have legal status or are informal. They need systems and user interfaces that make the writing of rules simpler and assist in checking that everything is correct.

Automated checking systems

Three code checking systems have been implemented against the IFC (Industry Foundation Classes) model (IAI, 2004):

- Solibri Model Checker
- ePlanCheck
- Design Check

Solibri Model Checker (www.solibri.com) was the first of the IFC-based systems to be released for industry use. It consists of a constraint set manager module that allows users to build 'rule sets' (Figure 8.4), the model checking/reporting engine itself (Figure 8.4) and a viewer that allows users to see which building component is causing a problem (Figure 8.5).

Solibri Model Checker was tested as a method of storing the requirements of AS1428.1, 'Design for Access and Mobility – General Requirements'. The software was easy to develop rules in, easy to use and produced excellent reports with the ability to embed 'snap shots' from the viewer of identified problems. The drawbacks were that there was no method for a user to access attributes and values that were not already exposed through the Solibri rule interface and it was not possible to write algorithms, for example to define an unobstructed path.

e-PlanCheck is the most widely known and publicised example of code checking since it was sponsored by the Singapore government (Khemlani, 2005). It covers the architectural and building services aspects of the Singapore code (Solihin, 2004b). It is based on the Express Data Model system (Solihin, 2004a).

Figure 8.5 Solibri viewer with identified problem.

Figure 8.4 Solibri constraint set manager and reporting engine.

In 2004, 92 per cent of the clauses in the architectural requirements were met and 77 per cent of the building services requirements. Most of these were removed due to the impracticality of checking rather than an inability to do so (Solihin, 2004b). While ePlanCheck provides 24 hours-a-day by 7-day-per-week access, it does not allow designers to define their own rules. It is consequently limited to checking against the provided requirements.

Checking for compliance

The Cooperative Research Centre for Construction Innovation (www.construction-innovation.info) launched the first stage of a two-stage project to develop code checking using AS1428.1, 'Design for Access and Mobility – General Requirements' as the requirements in 2001 and completed the second stage in 2005. The system uses the Express Data Model System like the Singaporean ePlanCheck system. The rules for checking compliance are encoded using Express-X, a constraint language defined as part of the ISO10303 (STEP) suite of standards.

Implementing code checking requires a combination of capable building modelling software and the rules to check against the stored data. The IFC model is commonly used as the means of encoding data due to the support by the major architectural CAD vendors and its compatibility with the STEP (ISO10303) standards.

Checking against some clauses is trivial. For example, AS1428.1 Clause 10.2.5 states that 'the toilet seat shall be of the full-round type, i.e. not open fronted.' While CAD systems do not normally support this level of detail, an extra property stating that the toilet seat was 'closed' was exported for toilet seats that had to be compliant with this clause.

Some clauses are unlikely to ever be fully supported from CAD. For example, clause 10.5.4 (c) states that 'floor tracks for sliding screens shall be flush across the shower recess opening'. It is difficult to imagine someone going to the trouble of building a building model to such detail.

Clause 5.1.2 states 'there shall be a continuous path of travel to and within any building to provide access to all required facilities.' Resolution of this clause requires an algorithmic approach. The method used to implement this rule was a recursive function that started at a complying entrance and then iterated through the spaces within the building model to find other spaces which were accessible (had a complying door or opening) from a space that is known to be accessible. Ramps and lifts were treated as complying openings to allow changes in level and storey to be accommodated (Figure 8.6).

The most difficult type of checking requires geometric reasoning. Figure 8.7 (from AS1428) shows a series of doors along a corridor with constraining dimensions. This type of situation is identified by finding spaces, locating the doors into the space, finding the direction of swing and then overlaying the 2D shapes to calculate the minimum distance between the door swings.

Implementing a code checking system

As mentioned earlier, some additional information must be added to the CAD model to allow full checking. These are passed through the IFC export interface as extension properties. These are not part of the official IFC model but can be handled as user-defined extensions. The IFC model allows the definition of 'proxy' objects. These are objects which are not part of the official IFC model but are understood in the context of a one-to-one information exchange. Some CAD

Figure 8.7 Door tolerances from AS1428.1.

Figure 8.6 Checking continuous routes.

systems also support library objects as proxies since they cannot determine their type explicitly.

A variation to the IFC model was defined that had additional objects that were necessary to cover the required scope. It also added the attributes that were referenced through the rules to the express definition of the object rather than to attached property sets to make the writing of rules more intuitive. This is called the 'internal model'. Processing a file consists of the following steps:

- export the model from CAD as an IFC file;
- read the model into EDM;

- convert all proxy objects of interest into objects of known types;
- map (convert) the IFC data to the internal model;
- run the rules as selected by the user and produce the reports.

Using design check

Using design check is a five stage process:

- select the rule set to be checked against;
- select the building model to check;
- select the rules or objects to check;
- run the check;
- examine and respond to the report.

The user can then make changes to the model if any are necessary and run Design Check again.

Figure 8.8 shows the code selection dialogue. Each code has a unique name under which the relevant rules are stored. The dialogue shows the two codes that are currently stored, AS1428 part 1, the current Australian standard for access for disabled access, and BCAD3, the draft provisions that are currently out for public comment as the next version of the disabled access provisions.

The user then selects the project to be checked from the project database using a standard file open dialogue. Checking can be performed based on either rules or on the relevant object types (Figures 8.9–8.12). When checking against rules the user can check against the entire code by selecting all clauses, or can select individual clauses or sub-clauses. The sub-clauses are accessed by drilling down through the tree view (Figure 8.9). The user can select as many rules as desired in whatever combinations are appropriate. This ability to choose relevant clauses supports the checking of designs at various stages of development. If, for example, the user has just finished laying out the spaces and doorways, access to all appropriate spaces can be checked by selecting the appropriate clause, without being overwhelmed in the reports by information about missing detail that has not been considered yet.

Once the 'Check Project' button is hit the rule checking is run and the results displayed (Figure 8.10). There are five possible results for each selected clause.

Figure 8.8 Selecting the desired code for checking.

Figure 8.9 Drilling down into clause hierarchy and selecting clauses.

'Passed' means that the clause is satisfied. 'Pass given' indicates that additional information has been added to the model to explain why a particular clause has not been met. This may be because a particular requirement has been met through an alternative method, as allowed under performance-based codes. 'Specification needed' indicates that information is required that is not supported by current architectural CAD systems. It is expected that this information would be covered in the project specification in some manner. References to a specification clause or other documents can be added and explained later. This facility allows the full 'design' to be checked even if a requirement is not supported by current technology. 'Model information missing' indicates that objects that are expected to be in the model are not there. The user can either add the information to the next iteration of the model or can add a reference to the project specification or other information (i.e. product manufacturer's brochure). This provides flexibility in the location of information. Within Design Check there is no requirement that all required information be entered in a particular manner. The final possible result, 'some issues' indicates that all of the required information is in the model, but does not comply with the clause requirements.