Making Automated Building Code Checking A Reality[1]

Charles S. Han, John Kunz, Kincho H. Law

Center for Integrated Facility Engineering
Stanford University
Stanford, CA 94305-4020

csh@galerkin.stanford.edu
kunz@cive.stanford.edu
law@cive.stanford.edu

1. Introduction

Currently, design and construction documents submitted to the governing body for permitting are checked manually against a continuously changing and increasingly complex set of building codes. The complexity and changing nature of codes leads to delays in both the design and construction processes. The designer must assess which codes are applicable to a given project as well as sort through potential ambiguity in the code provisions. An inspector must go through a similar process. In addition, inconsistencies in interpretation of a given section of the code may differ from inspector to inspector. The design checking and approval process can be a critical activity that prolongs the construction and delays the operation of a facility. Automating this process has the potential to alleviate both the delays and inconsistencies associated with manual checking by giving the designer and the permit-issuing body a consistent framework in which to apply and check codes.

During the Summer, 1996, we developed a proof-of-concept prototype demonstrating the feasibility of an online code-checking methodology. In this example, the program checks accessibility in a single accommodation toilet facility via the WWW in a client-server environment. On the client side, the user develops a plan of the toilet facility in AutoCAD. The user can at any time send this toilet facility design to a code-checker which is written in Java on a remote server. The server checker examines the data and sends back redlines (Figure 1) that are incorporated into the AutoCAD drawing and also constructs a Web page that has comments in one frame that are linked to the applicable code in another frame, in this case, portions of the California Access Compliance Source, Inc. (CalACS) Accessibility Standards Interpretive Manual (see Figure 2). This manual incorporates Title 24 and American Disability Act (ADA) Accessibility requirements. The client program then automatically pulls up this web page with a browser, and the user can peruse both the comments and the code.

Before automation of the code-checking process can become a reality, several technologies must be in place in order to facilitate the communication of design information to the correct official code-checking bodies. The World-Wide Web (WWW) has gained popularity as both an information gathering and reporting medium. Utilizing the World-Wide Web for a code-checking service makes sense except that there are still several standards of information transfer that need to be addressed.

The next two sections of this article are concerned with these emerging standards and technologies. These standards include a proposal for a formal description of a building model and a protocol for sending and retrieving data in a distributed computing environment. These technologies will also have an impact on other areas of facilities management. The subsequent sections of this article describe an approach to the problem of automating the code-checking process given that the standards are in place.

---

[1] A modified version of this paper appeared in the September/October 1997 issue of *Facility Management Journal*

2. A Building Model Standard

In order for automated code-checking to be successful, the program that does the analysis of a building design must understand the design information that is receiving from the source. There are two approaches to this problem. At one end of the spectrum, the designer can send a 2-D CAD drawing constructed with drawing primitives such as lines and arcs. It is then the responsibility of the analysis program to generate the functional information that is implied by a group of lines. For example, one group of arcs and lines must be identified as a door; another group might form a wall. Further, the program would then need to group these objects to generate spaces or rooms and also generate the function of these spaces. For example, one group of objects might make up the components of a theater; another might make up the components of a bathroom. Then the program would analyze these generated objects and spaces for code compliance.

The above approach describes the current state of CAD technology (one can, of course, group drawing primitives to create more meaningful objects, but designers often do not adhere to some universal standard). The lack of strict universal drawing standards poses major challenges to the generation of building components from drawing primitives.

At the other end of the content spectrum, a 3-D building model could containobjects that explicitly describe both functional requirements (e.g., spacesthat must support wheelchair access) and the design parameters (e.g.,dimensions, materials of walls). For example, as opposed to receiving a group of lines that implies a door, the program would receive an object that is a predefined symbolic representation of a door with attributes such as swing, material, fire-resistance, structural properties, handicapped-accessibility properties, and so on. In addition, spaces or rooms and their properties can be explicitly defined. For example, a room can be labeled as a commercial kitchen with all the specifications for the appliances, light fixtures, and mechanical systems explicitly defined and have the property that an egress path cannot pass through it; another room can be labeled as a public dining room with the egress path explicitly defined.

This second approach alleviates the need for the analysis program to interpret and generate much of the design information. However, other problems may arise with the assumption that the designer will correctly label all the functions of a building component or a space. For example, a designer may mistakenly label a space as not needing to be handicapped-accessible.

Clearly, for automated code-checking (and other types of analysis), there is a need for a standard model that provides more information than a collection of drawing primitives. Currently, there is an effort by the International Alliance of Interoperability (IAI), a consortium of CAD vendors and other AEC industry partners, to develop standards for a project model that enables interoperability between applications by different software vendors. A brief discussion of the IAI project model follows (for more information, see the IAI's web site at www.interoperability.com).

The IAI defines a set of objects called Industry Foundation Classes (IFCs) that have a hierarchical structure that conforms with object-oriented technology. This type of hierarchical classification scheme helps clarify our thinking of the nature of a specific object. For example, if we have a class called fixture, we can also have children classes of the fixture class that are more specific examples of fixtures such as lavatory and water closet. In the opposite direction, the fixture class might be a child class of a called "manufactured element." The advantage of this hierarchical structure is that attributes of a class are inherited by children classes. For example, if the fixture class has the attribute "(water) supply line," then its children classes lavatory and water closet also have this attribute. Classes can also have relationships with other classes

Let us examine some IFCs and their attributes. One class, IfcLayeredElement, is the parent class of three types of building components: IfcWall, IfcFloor, and IfcRoofSlab. Another class, IfcFillingElement, is the parent class of the building components IfcDoor and IfcWindow. All of these classes have common attributes (such as geometric characteristics) so it is not surprising that they have a common ancestor, IfcElement, in which these attributes are described. See Figure 3 for the partial IFC hierarchy.

Additionally, it is useful to describe a possible relationship between a door and a wall. An IfcDoor can be associated with another class IfcOpeningElement which in turn can be associated with another class IfcRelVoids which finally can be associated with an IfcWall. Defining these relationships provides a natural medium similar to the situation when we look at a plan that contains a door in a wall.

In addition to specifying the attributes and relationships among building components, the IAI also specifies two formats for exchange of data: static file exchange (one IFC-compliant application produces a file that can be read by another IFC-compliant application maintaining data integrity) and dynamic exchange of data through standard object-passing interfaces.

From a code-checking point of view, having a standard building model format allows the analysis program to concentrate on code issues as opposed to interpretation of drawing primitives. We may still need to assume certain things (such as if the designer creates a door object that has certain dimensions, and that those dimensions are correct), but these are the same assumptions that a building inspector usually makes when examining a set of design documents. Even though it is to the designer's advantage to accurately describe the relationships of the various building components, it is important for the code-checking program to be able to verify the relationships that are described in the IFC design data. For example, is this door really part of a specific wall system?

Other facilities management applications will greatly benefit from having a standardized way of describing building components. Having such a description of a project will be useful beyond the construction phase of the project and throughout a project's life cycle. As of this writing, the specifications for IFC Release 1.0 are complete. With IFC Release 2.0, other issues such as certain domain processes and usage scenarios will be developed. The Release 1.0 specifications are adequate to integrate the IFC project model in our code-checking work related to accessibility issues. The future development of IFCs will further provide a useful and universally recognized model.

3. An Object Broker on the Network

The World-Wide Web has greatly facilitated the use of the Internet and also been instrumental in generating the volume of information now available on the Internet. The Internet had been in existence long before the development of the World-Wide Web, but its use was limited to mainly educational, government, and a few commercial institutions. Presently, most of the information received through a browser is static (one can, of course do certain interactive things such as use a search engine or place an order). There are several ongoing efforts to make the web environment more dynamic including the development of a broadcast or push model in which information of some pre-selected topic can be sent to the user automatically. Another is a distributed object computing standard. There are several groups that are committed to the development of a standard for a distributed object computing environment. The benefits of such an environment are outlined in this section.

The Object Management Group (OMG) is a consortium that has developed the Common Object Request Broker Architecture (CORBA) (see www.omg.org) while Microsoft has its own initiative known as Distributed Component Object Model (DCOM) (see www.microsoft.com). Other common terms related to DCOM are ActiveX and OLE (Object Linking and Embedding). CORBA and DCOM have their relative strengths and weaknesses. Here, we will briefly discuss the advantages of using the distributed object computing environment in relation to the code-checking and other AEC-related problems. We will examine some of the aspects of CORBA since we are more familiar with that environment.

As stated in the introduction, determining which codes (and which version of a specific code) are applicable can be a complicated issue. It would be useful to submit a design (in our case, a building model) to a central service. The service would figure out which jurisdictions are applicable and send the building model to the appropriate jurisdictions to be automatically analyzed against their own set of codes.

The analysis would then be sent back to the designer through the central service. In essence, the central service described here would be acting as a broker.

In CORBA, the service described here is known as a Trading Service. When submitting the building model, certain constraints must be sent along: for example, perhaps all that is needed is the action desired (check for code compliance) and the location of the site of the proposed design (for example, Singapore). For such a trading service to work, the various jurisdictions would also have to register their services with the broker. The flexibility in this system is apparent especially if a jurisdiction changes its constraints. For example, a change in the local building code need not be published to all those interested (determining who the interested parties are is another complication). Only those who invoke the particular service will be notified.

One of the advantages of using such a broker is that the designer is shielded from how the information (objects) are sent from the designer to the appropriate jurisdictions and how the jurisdictions analyze the data. Conversely, when the appropriate jurisdictions send back their analysis of the design, they don't have to worry about how to route the information back if they go through the trading service. Of course, there has to be some kind of common language to describe the objects whether it is the building model or the code comments that come back. In CORBA, this common interface is known as the Interface Definition Language (IDL).

Obviously, such a distributed object system has implications in facilities management far beyond code-checking. Another service might involve sending the building model or some subset of the model, say all the light fixtures, to a central service to get the appropriate bulbs. Constraints could include things like price (or the lowest price), time for delivery, warranties, and so on. Again, this environment is dependent on the hardware stores in the region registering their inventory with the trading service, but this is analogous to the various forms of current-day advertising. Note also that here again, the need for a standard building model is necessary to communicate the type of facilities information we are requesting.

Integration of this type of distributed object environment with the World-Wide Web is the next logical step, and requests for CORBA services can be implemented on the client side using Java applets in browsers. Integrating Java in browser technology has given users slightly more power and interaction for information retrieval, but true integration of a distributed object environment will take full advantage of the power of the web.

4. Towards Automated Code-Checking

Now that we have described the necessary supporting components to make automated code-checking possible, we turn our attention to the actual code-checking problem. For the discussion, we will mainly concentrate on handicapped accessibility issues, specifically examining the provisions in the Americans with Disabilities Act Accessibility Guidelines (ADAAG) (see www.access-board.gov/bfdg/adaag.htm).

Given a symbolic building model that conforms to IAI IFC specifications, at the level of examining objects such as doors and water closets, we can model many of the provisions that pertain to these specific objects. For example, there are several provisions that pertain to the accessibility of a door including the width of a door opening and clearances on either side of the door that can be analyzed. Simple geometric tests can be applied to test a door for compliance.

Certain provisions cannot be directly captured. For example the section regarding the door hardware in the ADAAG states:

> **4.13.9* Door Hardware.** Handles, pulls, latches, locks, and other operating devices on accessible doors shall have a shape that is easy to grasp with one hand and does not

> require tight grasping, tight pinching, or twisting of the wrist to operate. Lever-operated mechanisms, push-type mechanisms, and U-shaped handles are acceptable designs…

How can phrases such as "tight pinching" or "twisting or the wrist" be modeled? There are several approaches to this problem. One way is to have an list of acceptable door hardware. As this list is dynamic, including such a list in the actual code is not practically possible. But given a distributed object environment as described in the previous section and a building model in which we can fully describe a building component (for example, Anderson door model x with Schlage lock y), a jurisdiction could have a dynamic list available for the automated code-checking process or any other related service for that matter.

Another way to approach this problem is to actually simulate the mechanics of the door hardware in question. Here, the building model would need to be detailed enough to convey properties such as torque, and again, the analysis program on the code-checking body side would also need to understand how to process these properties.

A more complex problem is deciding what code provisions are actually applicable to a given building component. For example, a door does not have to be accessible if it is not defined to be on the accessible path; exit signs are not required in certain structures and spaces. How can one identify what provisions are relevant for a given design? Again, there are several possible approaches. One way is to require the designer explicitly define code-relevant issues. For example, the designer can be required to specify the accessible path through a building; the designer is required to state what the function of a room or space is to determine whether or not an exit sign is required.

Requiring that all the code-relevant information be provided by the designer has several drawbacks. For one thing, putting the burden of addressing all the relevant code-issues on the designer defeats one of the purposes in automating the code-checking process: the volume and variety of regulations is difficult to keep up with and giving the responsibility of addressing local variations of the code on the jurisdiction alleviates this problem. Also, if the code changes, then it is possible that the information contained in the building model related to the code is no longer applicable. Finally, as mentioned earlier, in this approach, the automated code-checker assumes that the designer has accurately addressed the code issues and accurately labeled the functions of the rooms and spaces.

Another approach would be for the code-checking program to be able to determine what building components need to be accessible by analyzing the components in a room or space. For example, if the equipment in a room is all mechanical, then the program determines that the space is a mechanical room and accessibility is not required. Certain things would still need to be explicitly labeled such as the function of the building (residential, commercial, and so on), but this is true in the current manual code-checking process also. This approach is more difficult than the one where everything is explicitly defined, but it does offer more flexibility and power in the code-analysis process.

Finally, there are some issues that are difficult to analyze statically. If a door must be on an accessible path, then we can go ahead and check its necessary clearances, but since these are local checks, we cannot guarantee that in getting from one room to another, even if the individual doors meet the code, we can actually get to these doors. The ADAAG specifically states:

> **4.3.3 Width.** The minimum clear width of an accessible route shall be 36 in (915 mm) except at doors (see 4.13.5 and 4.13.6). If a person in a wheelchair must make a turn around an obstruction, the minimum clear width of the accessible route shall be as shown in Fig. 7(a) and (b).

Here, we take a different approach to analyze the building model for accessibility by simulating whether a wheelchair successfully moves through the space. Using motion planning, a research area in robotics, such a simulation is possible (see www-leland.stanford.edu/~wwwcsh/research/mp/).

Pursuing the simulation approach to code-checking, it may be possible in the near future to determine code-compliance in a more general and more powerful way. For example, instead of complying with a set of prescribed measurements in order to ensure accessibility (again, looking at a door, there are several including minimum opening width, pull-side and push-side clearance, strike-side clearance, and so on), it may be possible to simulate a robot in a wheelchair interacting with a door. If this robot can successfully go through a door, then even if one of the prescribed measurements is not met, one may decide that the door does meet the accessibility requirements.

Similarly, if it is possible to simulate a group of people trying to exit a space, prescribed measurements such as exits needing to be less than half the diagonal distance of a room apart may not be required. Instead, if the simulation determines that all the people can get out of a space before some critical event such as the structure filling with smoke, then the space complies with egress requirements.

Currently, there are many obstacles to this general approach. Even with the simple example of a robot in a wheelchair trying to go through a door, the computational power to perform such a simulation exceeds current capabilities. Additionally, in the egress example, capturing the exact behavior of people in a building (or accurately modeling the burning building itself) is not a trivial task. Our research approach examines ways to incorporate simple simulations (such as determining if an accessibility path exists) with rules that capture the prescriptive provisions of components along with methods to determine the relevance of these provisions on a specific building component.

5. Examples

Automated code-checking or standards analysis and compliance has been an active area of research since the 1960s. With the computational and communication infrastructure are beginning to be developed, we can start to realistically test our code-checking models. An overview of the areas of research that we are working on can be seen at our website at www-leland.stanford.edu/~wwwcsh/research.html. We are continuing to work on a client/server web-based interface for an automated code-checker (see Figure 4 and www-leland.stanford.edu/~wwwcsh/research/abcServer/abc.html). Note that the browser needs to be VRML-compliant to see the building model in one of the frames (see vrml.sgi.com).

This web page was generated by our prototype ADAAG code-checker that analyzes an IAI IFC Release 1.0-compliant building model. There are links from the redlines of the VRML frame to the code comments, and then links from the code comments to the provisions of the actual ADAAG code. See Figure 5 for the client CAD model and Figure 6 for the generated web page.

This interface stresses our desire to have a platform-independent implementation of our prototype. Any IFC-compliant CAD system will generate the building model (at our website, one can download an AutoCAD file with an AutoLISP and Java utility that can generate and send an abbreviated version of an IFC Release 1.0-compliant file to our code-check server), and the information can be examined with a VRML-compliant web browser. Also at our website, there are examples of some VRML animation that can be generated given a path that is created by our motion planning simulation (see Figure 7 for a snapshot of the wheelchair simulation). This simulation module will be incorporated in our code-checking prototype in the near future. A distributed object environment will take full advantage of the power of the web.

6. Summary

With the increasing time pressures of the realization of a building project, automated code checking is an important step in the design and building process. For automated code checking to be feasible, several standards and technologies need to be in place including the standardization of a building model and a

communications model for a distributed object networking environment that will facilitate not only the transfer of this model, but the routing of the model to the relevant parties. This article examines some of the relevant standards and technologies and our research approach to the automated code checking problem.

7. Acknowledgements

This work has been supported by a Center for Integrated Facility Engineering seed grant. CIFE sponsors seed research projects in three research areas: Automation, Integration, and Management of Technology.

8. References

California Access Compliance Source, Inc. (CalACS) (1994). *Accessibility Standards Interpretive Manual*, Builder's Book, Inc., Bookstore, Canoga Park, CA.

Fenves, S.J., Garrett, J.H, Kiliccote, H., Law, K.H., Reed, K.A. (1995) "Computer Representations of Design Standards and Building Codes: U.S. Perspective," *The International Journal of Construction Information Technology*, University of Salford, Salford, U.K.

International Conference of Building Officials (ICBO) (1994). *Uniform Building Code*, Whittier, CA.

International Conference of Building Officials (ICBO) (1995). *Handbook to the Uniform Building Code*, Whittier, CA.

Latombe, Jean-Claude (1991). *Robot Motion Planning*, Kluwer Academic Publishers, Norwell, MA.

Vogel Andreas, and Keith Duddy (1997). *Java Programming with CORBA*, John Wiley and Sons, Inc., New York, NY.

Automated Building code Compliance Checking Where is it at? Article. Full-text available.Â Automated regulatory compliance checking requires automated extraction of requirements from regulatory textual documents and their formalization in a computer-processable rule representation. Such information extraction (IE) is a challenging task that requires complex analysis and processing of text. Natural language processing (NLP) aims to enable computers to process natural language text in a human-like manner.Â Making Automated Building Code Checking A Reality1. Article. Jan 1997.