An XML Framework for Simulation and Analysis of Buildings

Filiz Ozel, Robert Pahle, and Manu Juyal

An XML Framework for Simulation and Analysis of Buildings

Filiz Ozel, Robert Pahle, and Manu Juyal

This study focuses on the problem of how to structure spatial and component based building data with the intention to use it in the simulation and analysis of the performance of buildings. Special attention was paid to the interoperability and optimization of the resulting data files. The study builds its investigation onto XML (Extensible Markup Language) data modeling framework. The authors have studied different ways of arranging building information in XML format for effective data storage and have developed a data modeling framework called bmXML for buildings. Initial results are two-fold: a VBA application was developed to create the appropriate building model in AutoCAD with the intention to write building data in bmXML format, and a JAVA application to view the file thus created. This paper primarily focuses on the former, i.e. the AutoCAD application and the bmXML format.

1. Introduction: Industry Problems in Data Exchange

With the increase in the use of electronic media for design information sharing, an issue that has become urgent in architecture, engineering and construction industry is the seamless sharing of such information between different stakeholders. Many software developers have their own proprietary data formats to store geometric as well as attribute data, and this has become an important concern when the multitude of formats started becoming a hindrance to data sharing. In short, there is no standard format that satisfies the large number of information exchange needs for Architectural Geometry, Project Management, Building Analysis, etc.

In the past, some of the data formats used in AEC software have become quasi standards such as AutoDesk's Drawing Exchange Format (DXF). While DXF remains to be a graphic data exchange format, it has not been extended to domain specific data such those for architectural elements. For other formats, typically there are many problems. For instance, copyrights may be owned by a specific software company, thus every use may have to be licensed. The implementation of methods to read and write such files is also difficult, since data is often encrypted and not in plain text, which increases the challenge of parsing such files.

Within this context, researchers have proposed and implemented different frameworks. The Industry Alliance for Interoperability (IAI) and STEP are among the most widely coordinated efforts in data modeling and standardization [1,2,3,4,5]. Architectural data modeling requires geometric modeling of the built environment, where a number of very different geometries as well as their spatial relationship to each other must be accommodated [6,7]. Furthermore, extensibility, i.e. adaptability of the existing format to new data requirements, is an important requisite characteristic of such data exchange formats.

Extensible Markup Language (XML) is emerging as a new paradigm in data modeling, especially as it relates to e-Business applications [8] as well as Internet based collaborative design support systems. XML, like HTML (Hypertext Markup Language), is a subset of the Standard Generalized Markup Language (SGML) that is defined in ISO 8879. HTML is limited to a set of predetermined tags that can only be parsed in predefined ways, whereas XML provides the flexibility to create one's own tags based on the specific needs of the application being developed. Furthermore, the XML-Standard partially describes how an application processes such files [9]. This technology is also being applied to the modeling of data related to architectural environments by the IAI [10].

In this research, authors have focused on structuring spatial and component based building data with the intention to use it in the simulation and analysis of the performance of buildings. Different ways of arranging building information in XML format was studied in order to optimize data storage. As a result, a data modeling framework called bmXML was created.

This article focuses on this framework as well as on the creation of data files in bmXML format using AutoCAD's Visual Basic for Applications programming extension. The application captures the geometry of building spaces and components, as well as the relationship between these objects. Another application, developed in Java, parses the XML file using SAX technology, and creates related object classes in the JAVA application. The latter will be covered in subsequent papers

2. Related work

2.1. XML frameworks for the built environment

XML is a combination of rules, guidelines and conventions, which allow the creation of text files for structured data. This structured data could be an address book, configuration parameters for technical drawings or for building components. Programs, which use the data, could also save this information in binary format. But if it is saved in text format, one could view and change the data without having to use the program that generated the data file in the first place. The usual difficulties like insufficient extensibility, missing support for multiple languages or platform dependency are not issues with XML files. [11] Another reason to choose XML for data modelling is its wide availability as well as the existence of many development tools, which can be used with it. [12,13,14,15]

Unlike HTML, XML does not predefine what a certain tag means. XML uses tags only to organize the data and leaves the interpretation to the application that uses the data. Although this is an advantage in terms of flexibility, the rules in XML must now be much stricter than in HTML. Strict validation of XML data files is necessary before it can be parsed by an application.

2.2. aecXML

During the past few years, there have been numerous efforts to implement standards that describe architectural, engineering and construction (AEC) data in XML format. Among these, aecXML is an organized collection of architectural, engineering and construction information that is intended to be used via the Internet. Initial work and the publication of the framework were done by Bentley Systems in August 1999. It is designed for all the non-graphic data involved in the construction industries, and is described to have a place alongside the Industry Foundation Classes defined by the IAI. In the mission statement for aecXML, IAI states that "It is intended to be used as an XML namespace and to facilitate the exchange of AEC data on the Internet" [10]. Non-graphic data for AEC projects was described to come from "resources such as projects, documents, materials, parts, organizations, professionals or activities such as proposals, design, estimating, scheduling and construction" [ibid].

Standardization efforts in building design can be grouped into two categories. One type of effort aims to describe the process of design and construction (as with aecXML), while the other type aims to describe the geometry of architectural objects and their relationship(s) to each other (Table 1). What is needed for simulation and analysis purposes is clearly in the latter category.

2.3. ifcXML

The ifcXML by IAI, which is an XML frame work that includes IFC (Industry Foundation Class) model, and BLIS-XML [16], which is an XML framework to encode EXPRESS based information, fall under the category of systems that are designed to store the information about a building, its geometry and the relationship between building parts. These frameworks include extensive abilities to model information used in the AEC industry. The intention behind ifc and ifcXML is stated by the Alliance as "to develop a standard universal framework to enable and encourage information sharing and interoperability throughout all phases of the whole building life cycle" [1]. This implies a highly complex and extensive data modeling format, since all information sources and all AEC domains need to be taken into account.

On the other hand, BlisXML is intended to be a general object-relationship builder, so that complex relations can be created using this method of storing information irrespective of data semantics or context. Individual pieces of data can have a relationship to each other (akin to a relational database system) without having to identify a semantic framework, but parsing of such a data format can be very complex. The provision of a numeric identifier for each tag does not provide any semantic information regarding the underlying nature of the data set. Thus metadata that could have been provided by semantically loaded tags is lost when information is saved in BlisXML format.

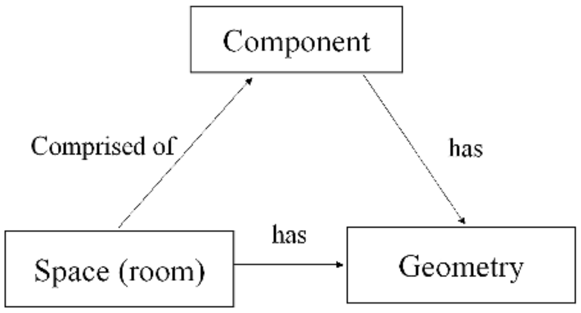
▼ Table 1. Some XML Frameworks for the AEC sector

XML frameworks to store the design process or workflow			**XML frameworks to store physical environment information**		
aecXML	-	XML framework for Architecture, Engineering and Construction (AEC) industry http:// www.aecxml.org/	ifcXML	-	XML framework, for building information http://www.iai-international.org/
ebXML	-	electronic business XML http://www.ebxml.org/	blisXML	-	framework made up from IFC objects http://www.blis-project.org/
citeXML	-	Centre for e-Business in Construction XML is Similar to AEC http://www.cite.org.uk/	EnerXML/	-	framework for building EnXML energy simulation, still in development http://www.hvac.okstate.edu/pdfs/bs01/BS01_0257_262.pdf
bcXML	-	building construction XML http://www.bcxml.net/	DesignXML	-	ACAD XML framework to represent geometry as in DXF files http://www.designxml.org/
LandXML	-	sub part of aecXML, mainly for process information for surveying http://www.landxml.org/	bmXML	-	XML framework, which focuses on simulation and performance analysis of buildings

The complexity and the ineffectiveness of the frameworks listed above led the authors of this article to consider developing a very simple but extensible framework that can be easily used by multiple applications in building simulation and analysis; as a result, bmXML was developed with mainly three types of object tags: space, component and geometry. What was needed was not a design process modeler, but more of a design product modeler, i.e. a method to define the final architectural artifact. This model will be discussed in more detail in the coming sections.

2.4. bmXML framework

During the development process, designing an XML structure that best captures the nature of the building data has been the most challenging part. The main issue can be summarized as the question of how to capture the relationship of building components to individual spaces, which in turn can have a relationship to one or more components as well as to multiple geometric entities (Fig. 1). Support for multiple geometries/topologies in 2D and/or 3D format for individual elements was found to be critical, since different applications require different levels of specificity in defining component geometry as well as space (room) geometry. For example, a travel algorithm may only require a 2-dimensional geometric data set for rooms, while life cycle analysis software may require 3D massing and detailed 3D geometric modeling of every single component in a building.

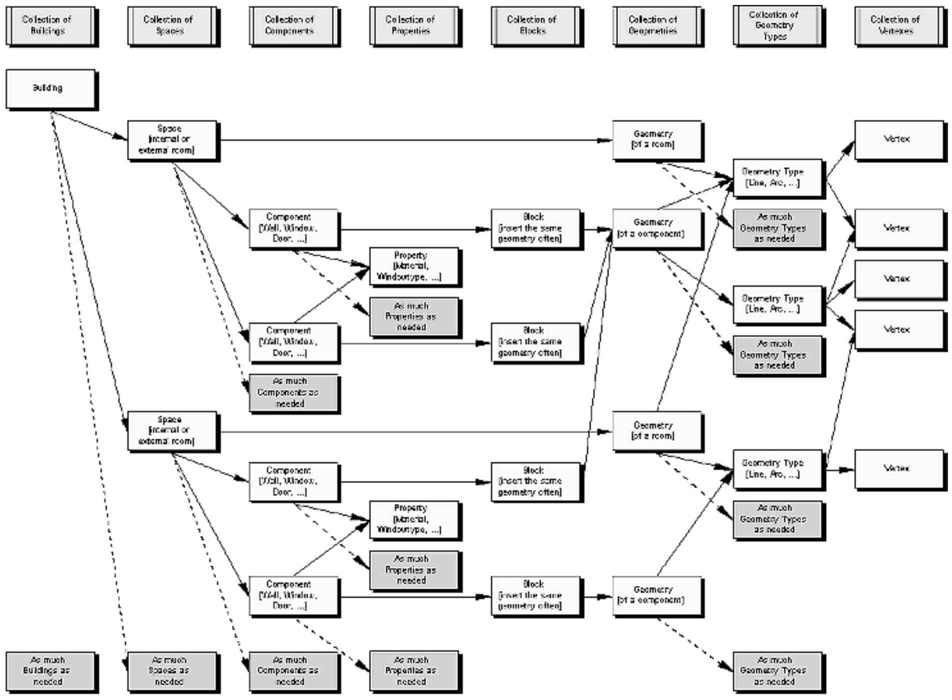


◀ Figure 1. Conceptual structure of the building definition in XML

The difficulty with the structure seen in Fig. 1 is that spaces (rooms) must, at times, be defined through geometry that is independent of the geometry of its individual components; while at other times a more detailed rendition of the space (room) geometry must be generated through the description of its individual architectural components. Although one might argue that the latter should always be the case, complete dependence of space (room) geometry on the geometry of the individual components eliminates the possibility of this data model supporting software for early design phase. At this phase, design decisions are still schematic and component geometries are yet vague. The same is also true for the data modeling needs of the analysis and simulation software at the schematic phase of the design process.

At first glance, it seems like such a space-component-geometry trilogy can best be supported by a hierarchic data structure, but due to the need to define geometric alternatives for each space or component as well as to avoid inevitable repetitions of data units in hierarchic structures, other data modeling methods were explored.

In addition to the hierarchic structures that are native to XML technology, a pointer system (network model) was tested and will be described in this paper. The authors also developed Java code for the sole purpose of testing these two different models, while also considering real world scenarios for understanding the nature of real world objects in buildings.



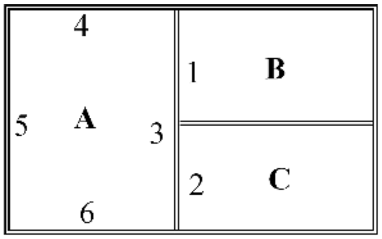
Collection of Buildings
Collection of Spaces
Collection of Components
Collection of Properties
Collection of Blocks
Collection of Geometries
Collection of Geometry Types
Collection of Vertexes
Building
Space [internal or external room]
Geometry [of a room]
Vertex
Geometry Type [Line, Arc, ...]
Component [Wall, Window, Door, ...]
Block [insert the same geometry often]
Geometry [of a component]
As much Geometry Types as needed
Vertex
Property [Material, Windowtype, ...]
Vertex
As much Properties as needed
Geometry Type [Line, Arc, ...]
Component [Wall, Window, Door, ...]
Block [insert the same geometry often]
Vertex
As much Geometry Types as needed
As much Components as needed
Space [internal or external room]
Geometry [of a room]
Geometry Type [Line, Arc, ...]
Vertex
Component [Wall, Window, Door, ...]
Block [insert the same geometry often]
As much Geometry Types as needed
Property [Material, Windowtype, ...]
As much Properties as needed
Component [Wall, Window, Door, ...]
Block [insert the same geometry often]
Geometry [of a component]
As much Buildings as needed
As much Spaces as needed
As much Components as needed
As much Properties as needed
As much Geometry Types as needed

► Figure 2 Object model of bmXML

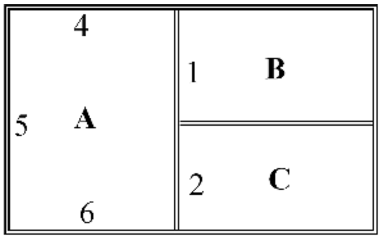
3. BModel

3.1. BModel XML Data Structure

The object model of bmXML can be seen in Fig. 2. Major object tags in bmXML framework are SPACE, COMPONENT and GEOMETRY. "Spaces" can be described as an area or – if we look at a real building – as a room, terrace or a carport. Buildings are, of course, comprised of entities other than just "spaces". Architectural components such as walls, doors or windows are part of a building. In bmXML these building elements are called "components" and the COMPONENT tag is used to describe them. To define the relationship between "spaces" and "components", a pointer called *"linkedcomponent"* tag is included as a sub tag of *"space"*. This tag can be repeated as often as necessary to describe the relationship between components and spaces. The "linkedcomponent" tag refers through the *"componentID"* attribute to a specific *"component"*. This is necessary in describing "components" that are shared among different "spaces", such as two adjacent rooms that share a wall or a door.



◀ Figure 3a. Spaces and shared components with redundancy



◀ Figure 3b. Spaces and shared components without redundancy

However, there is an important decision to be made for allocating components to rooms, much more so if the component is a shared wall. While allocating a component, we need to make sure that it cannot be further subdivided, e.g., in fig.3a, wall 3 must be divided into wall 1 and wall 2. This will guard the model against redundancy of information. In fig 3a and fig 3b, room A shares walls 1 and 2 with the rooms B and C. Room A can also be said to have a wall 3. But this will cause redundancy, because wall 1 and 2 together also comprise wall 3. Moreover, if plan is modified and anyone of the walls is changed, then the change will not be reflected in other walls. It will not only create redundancy but also ambiguity. To avoid any such redundancy and ambiguity we consider a space to consist of atomic components, i.e. components cannot be further sub-divided, although they can include other types of sub-components.

The following example demonstrates the data model for a "space" – in this case a room – with some "components"- doors and how they are connected with each other.

```
<Space id="1" type="Room" geometryid="4">
 <LinkedComponent componentid="1">
 <LinkedComponent componentid="7">
 <LinkedComponent componentid="9">
</Space>
```

As seen above, the space-tag has three attributes that describe a unique *id*, the type of space room/terrace/etc. and a *geometryid* that references a specific geometry. The *"component"* tag itself is there to describe elements such as walls, doors, or windows related to a specific space. For example the room above has a door *"component"* as defined below (*componentid* of the room references the *id* of the door) :

```
<Component id=″1″ type=″Door″ geoblockid=″4″>
 <LinkedMaterial materialid=″2″>
 <DoorType type=″Sliding″>
</Component>
```

Component description works in the same way as space description, except with an additional attribute *"geoblockid"*, which refers to the geometric block the component is comprised of. Additional tags can be included like *LinkedMaterial*, which refers to a material description, or *DoorType*, which defines more exactly how a door opens. The information can be further expanded by other researchers to meet other data requirements. The *"geoblockid"* is important for storing geometries of objects, such as those for doors, windows, furniture, etc. only once and then for using it multiple times throughout the building model. The following example defines a geoblock for the door component seen above (*geoblockid* of the door references the *id* of the geblock as seen below):

```
<GeoBlock id=″4″ insertPoint=″12″ angle=″0″
scaleX=″0.8″ scaleY=″0.8″ scaleZ=″0.8″ mirror=″yes″
geometryid=″7″/>
```

The *geometryid* attribute with a value of "7" in Geoblock refers to the *id* of the geometry as shown in the example below.

```
<Geometry id=″7″>
 <Line startVertex=″0″ endVertex=″42″/>
 <Arc centerVertex=″0″ startVertex=″42″
 endVertex=″43″/>
</Geometry>
```

This structure can be extended to curves or other tags to describe three-dimensional geometries.

Because XML technology is extensible, it will be possible for other researchers to add additional tags to the bmXML instance file with the intention to model objects and processes that are related to domain-specific areas such as energy analysis, fire analysis, code compliance checking. The extensibility of bmXML can be demonstrated by modifying the material description for the door component with an id of 1 that was already described above. This component was made up of the material with material id "2". The data provided below describes the details of the material with this materialid.

```
<Material id="2" type="Wood" />
  <Thickness>1.0</Thickness>
</Material>
```

So far this data model only gives a general description of the material of the door, which is good mainly for visualization purpose. The same code can be extended for the purpose of energy analysis by including R-Value of the material as an additional tag as in the code provided below.

```
<Material id="2" type="Wood" />
  <Thickness>1.0</Thickness>
  <Rvalue>1.0</Rvalue>  ------->  new tag for energy analysis
</Material>
```

This code can be further extended to include fire-ratings also, for fire analysis.

```
<Material id="2" type="Wood" />
  <Thickness>1.0</Thickness>
  <Rvalue>1.0</Rvalue>
  <Frating>1.0</Frating>  ------->  new tag for fire analysis
</Material>
```

3.2. Referencing vs. Nesting

One of the main topics in an XML data structure is whether to include tags in a nested manner which requires the inclusion of some objects more than once in order to establish the relationship between the parent and the child objects (such as interior walls must be repeated for every room they border) or to store each piece of information only once and then to reference it through the use of "pointers" implemented as attributes in other tags. As the description of the bmXML data structure above demonstrates, BModel uses a reference (or pointer) system to avoid the storing of redundant information.

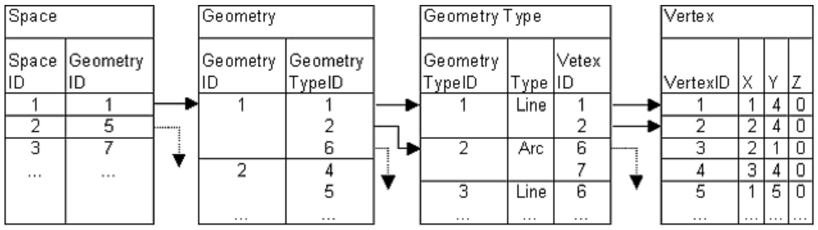
XML-developers have been discussing the advantages and disadvantages of storing data in a normalized fashion as per the mathematical theory of relational databases. [17]. On this issue, no consensus has been reached yet. The main argument for controlled redundant content is the easier graspable structure that allows a simpler overview throughout a generated document. It is also argued that large documents will be created only by automated programs and not in a manual manner, thus data redundancy is not a problem [18].

Nevertheless, only reference (or pointer) systems make it possible to really avoid redundant information. During the development process, the authors tested these versions to see how many tags were needed. Table 2 shows a comparison of how many attributes and tags are needed to describe a simple one-bedroom apartment including a terrace. Although smaller in size, the gain due to this smaller size in the reference system is not that great in comparison to the nested version. On the other hand, the advantages in representing the dependencies and relationships between parts of a building for shared objects such as a wall or a door between two connected rooms can be significant in the reference system. It eliminates the need for the simulation and analysis software to search for matching components at run time by using "multiple data comparisons" within the nested system. Another argument in support of the reference (or pointer) system is the potential to reuse a single geometric entity throughout the model in a repetitive fashion, as demonstrated in the previous section.

► Table 2. Use of Tags and Attributes in nested and referenced versions of a simple apartment.

	Nesting	Referencing
Tags	412	378
Attributes	777	723

Furthermore, numerous examples of how the ID attribute has been used as a pointer in bmXML can be seen in the previous section of this paper. Fig.4 demonstrates the use of ID for defining the geometry of a space in bmXML structure.



Space

Space ID	Geometry ID
1	1
2	5
3	7
...	...

Geometry

Geometry ID	Geometry TypeID
1	1
	2
	6
2	4
	5
...	...

Geometry Type

Geometry TypeID	Type	Vetex ID
1	Line	1
		2
2	Arc	6
		7
3	Line	6
...	...	...

Vertex

VertexID	X	Y	Z
1	1	4	0
2	2	4	0
3	2	1	0
4	3	4	0
5	1	5	0
...	...	...	...

► Figure 4. Referencing by using id

4. Applications

4.1. Creating the Structure with AutoCAD

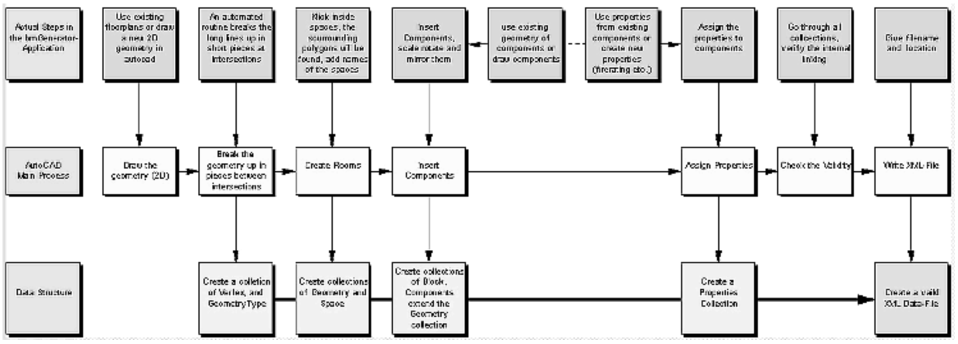
BModel intends to serve as a mechanism for storing component and space based geometric information as well as non-spatial information for simulation purposes. The AutoCAD application, which is currently being developed, serves thereby as a starting point in creating the basic simulation environment.

AutoCAD's Visual Basic for Applications programming extension was used to create the data files in bmXML format (bmGenerator). For this, not only appropriate object classes, such as Space and Components were developed, but also a mechanism for attaching different geometries to each object through the use of Collections was implemented. The Geometry Collection can hold any type of geometry that belongs to a given Space or a component. (See Fig. 2) This allowed the creation of bmXML file structure through method calls to the instances of individual objects in the VBA application.

The class methods to write the geometry in XML format allow the handling of different geometry types. Obviously, counter methods to read the geometry must be written in the Java application, since the intention is to create the geometry through AutoCAD and to process this data set through Java parsing.

Typically, each Room object, (i.e. the Space object in bmXML) has at least two public Draw methods, one that uses the geometry of its components, and the other that directly uses the geometry that resides in the Geometry collection of the Room. Therefore, a cascading effect can be created by only processing the geometry of an object through the geometries of its sub components, or a direct geometry can be defined in those cases where components are not detailed yet. This was seen necessary in order to create a scalable and extensible object model, i.e. a model that can handle space and component geometries at different levels of completeness.

In order to make the creation of the bmXML framework easier, AutoCAD application bmGenerator provides the options to use existing two-dimensional drawings or to create new geometry from scratch to define the "spatial" structure. The ease of use was an important concern in the design of the user interface. Therefore, the process of clicking into a room to select the geometry that belongs to that room was implemented. The system automatically locates all of the bounding walls and creates the room geometry. Additional components and properties (such as fire rating) can be added later. Finally the created model is validated and written as a bmXML file using the Geometry collection for each "space" instance. (Figure 5)


Actual Steps in the bmGenerator-Application
Use existing floorplans or draw a new 2D geometry in autocad
An automated routine breaks the long lines up in short pieces at intersections
Klick inside spaces, the surrounding polygons will be found, add names of the spaces
Insert Components, scale rotate and mirror them
use existing geometry of components or draw components
Use properties from existing components or create new properties (firerating etc.)
Assign the properties to components
Go through all collections, verify the internal linking
Give filename and location
AutoCAD Main Process
Draw the geometry (2D)
Break the geometry up in pieces between intersections
Create Rooms
Insert Components
Assign Properties
Check the Validity
Write XML File
Data Structure
Create a colletion of Vertex, and GeometryType
Create collections of Geometry and Space
Create collections of Block, Components extend the Geometry collection
Create a Properties Collection
Create a valid XML Data-File

◄ Figure 5. AutoCAD Process Model Diagram (bmGenerator)

5. Conclusion and Future Work

BModel is an easily useable and extendable framework, which focuses on creating a building definition for simulation and building performance purposes. It uses a referencing system to store dependencies among the elements of a building model. Two separate applications were developed to generate and to parse the bmXML files. An effort was made to standardize the information needed for basic building simulation purposes. Other elements can be later added to bmXML structure to make it extensible for different kinds of simulation and analysis purposes.

Among these, an important one will be <objectsContained> tag, since many simulation efforts such as egress, ADA as well as life cycle analysis of buildings require that all objects located in a given space be accounted for. While "Type" attribute will identify the type of an object, whether it is furniture, human being or goods, "ContainedIn" attribute will be a pointer to the space the object is located in. The latter will reduce the processing time considerably while the simulation such as egress simulation is running, since the need to find the "container" of an object by polling the spatial properties of all containers (i.e. spaces) is eliminated by including a readily available reference to the "container".

The first phase of bmXML development aimed at capturing the geometric information needed for life safety simulation, but by the virtue of its structure as spaces and components, it can also be used to store the information needed for various other kinds of building simulations

Next step will involve the development and implementation of more sophisticated procedures to create and save objects and properties in bmXML format for simulation purposes and to parse this data set for fire and energy simulation applications.

Acknowledgements

The authors would like to thank the PhD Program in Environmental Design and Planning, Arizona State University for their support.

References

1. *International Alliance for Interoperability: 1998*, Industry Alliance for Interoperability; http://www.iai-na.org/.
2. Ozel, F., Data Modeling Needs of Life Safety Code Compliance Applications, in: *Proceedings of the 1992 Conference of the Association for Computer Aided Design in Architecture (ACADIA'92)*, Charleston, SC, 1992.
3. Shaviv, E. and Peleg, U. J., An Integrated KB-CAAD System for the Design of Solar and Low Energy Buildings, in: G. Schmidt, ed., *Proceedings of CAAD Futures '91 Conference*, Zurich, Switzerland, 1991.
4. STEP: "Standard for the Exchange of Product Data", ISO CD 10303, 1993.
5. Eastman, C., *Building Product Models: Computer Environments Supporting Design and Construction*, CRC Press, Boca Raton FL, 1999.
6. Ozel, F., Modeling in the Simulation of Fire/Smoke Spread in Buildings, in: *Second International Conference of the Latin-American Society in Digital Graphics*, Sociedad Iberoamericana de Grafica Digital (SIGRADI'98), Universidad Nacional de Mar del Plata, Argentina, 1998.
7. Turner, J., An Application of Geometric Modeling and Ray Tracing to the Visual and Acoustical Analysis of a Municipal Open-air Auditorium, in: *Proceedings of the 1990 Conference of the Association for Computer Aided Design in Architecture (ACADIA'90)*, pp. 173-185.

8. Linthicum, D. S., *Understanding Data-Oriented B2B Application Integration*, Addison Wesley, New York, 2001, pp. 33-49.

9. W3C.org, *Extensible Markup Language (XML) 1.0 (Second Edition)*, Retrieved 2002.10.15, from http://www.w3.org/TR/REC-xml.

10. *International Alliance for Interoperability: 2002*, aecXML, Retrieved 2003.5.26, from http://www.iai-na.org/aecxml/mission.php.

11. W3C.org, *Homepage World Wide Web Consortium*, Retrieved 2002.10.15, from http://www.w3.org/.

12. Birbeck, M., *Professional XML*, 2nd ed, Wrox Press, Birmingham, UK. 2001, xiii, 1269.

13. Deitel, H. M., Deitel, P. J., Nieto, T. R., Lin, T. and Sadhu, P., *XML : how to program*, How to program series., Prentice Hall, Upper Saddle River, N.J. 2001, xxxviii, 934.

14. Goldfarb, C. F., *The Roots of SGML*, 1996, Retrieved 2002.10.03, 2002, from http://www.sgmlsource.com/history/roots.htm.

15. Goldfarb, C. F. and Prescod, P., *Charles F. Goldfarb's XML handbook*, Charles F. Goldfarb definitive XML series., 4th ed, Prentice Hall, Upper Saddle River, NJ. 2002, lvii, 1147.

16. Hietanen, J., *BLIS-XML,* 2000, Retrieved 2002.10.07, 2002, from http://www.blis-project.org/BLIS_XML/

17. Silberschatz, A., Korth, H. F. and Sudarshan, S.:, *Database Systems Concepts*, 4 ed, McGraw Hill, New York, 2001.

18. Jeckle, M., *Entwurf von XML-Sprachen, XML Spektrum im Java Spektrum*, 2000, Vol. 6, No. 28.

Filiz Ozel, Robert Pahle, and Manu Juyal, Arizona State University, College of Architecture And Environmental Design, Tempe, AZ, 85287-1605, USA

Ozel@Asu.Edu,
Robert.Pahle@Asu.Edu,
Manu.Juyal@Asu.Edu