Formalism for Detecting Version Differences in Data Models

Hongjun Wang¹; Burcu Akinci²; and James H. Garrett Jr.³

Abstract: In the architecture/engineering/construction (AEC) industry, a large number of data models (e.g., data exchange standards and task-specific data models) have been created and utilized to represent and exchange data in software packages. To meet the ever-expanding requirements for modeling real world information, the data models need to be updated frequently. Accordingly, those who need to implement these data models in their AEC-related software which often requires that they possess civil engineering domain knowledge, have to change their existing implementations for compliance with these models to account for the latest update. Before adopting changes of such data models, those developers working at AEC-related software companies must precisely identify which parts of the data models have been modified in a new release. Given the growing scale and complexity of today's data models involved in the AEC domain, identification of differences in two versions of a data model is a time-consuming and error-prone process, when performed manually. A semiautomated approach that identifies the differences in two versions of a data model could enable a rapid update of existing implementations of the model in AEC-related software. Due to the likelihood of having some commonality between the two versions of a model, it is possible to automatically identify version differences accurately. In this paper, we present an approach for detecting the differences between two releases of the same data model accurately and efficiently. This approach incorporates taxonomy for describing possible differences between two versions of a data model and provides a way to classify these differences. A prototype is implemented and used to validate the approach with the recent releases of some real world data models. The approach developed in this paper can help AEC-related software developers adopt and implement data models in their software systems.


CE Database subject headings: Information management; Construction industry; Computer software.

Introduction

For decades, the architecture/engineering/construction (AEC) industry has deployed hundreds of software systems to support various civil engineering tasks like architecture design, construction management, and facilities management (iCivilEngineering 2006). Accordingly a large number of private data models have been developed for these software systems to represent task-specific information (e.g., geometric data, HVAC design, and project information). In addition, to enable the data exchange between various AEC software packages, several public data exchange standards have been created in the AEC industry, including the Industry Foundation Classes (IFC) (IAI 2003a), ifcXML (IAI 2003b), CIS/2 (AISC 2003), AEX (FIATECH 2004), etc. For example, the IFC data exchange standard is an effort initiated by the International Alliance for Interoperability (IAI) and well accepted by dozens of AEC software systems to exchange data through different phases of AEC projects (IAI 2003a; Steinmann 2004).

Generally, most of the recently developed data models organize their contents in an object-oriented manner, incorporating both classes and attributes. A class is created to define real world objects and concepts, while an attribute is used to define or describe the associated class. However, it is not realistic to expect that a data model, in its initial release, can cover all possible data and requirements of parties in a particular domain. When a version of a data model is published, it is tested by real world engineering cases, resulting in generation of a set of comments and feedback on how it should be improved. Two major requirements drive modifications in a data model. One is the requirement of having new features to extend its coverage and enhance its functionality. The other is making minor modifications to address some problems or bugs identified in the existing releases. For example, Table 1 shows the major releases of the IFC specifications and how the data schema was changed from one release to another. The first release was introduced in 1997. Initially, it mainly focused on exchanging component descriptions, notably their geometric data, among CAD software packages. Since then, the IAI has frequently expanded the specification, covering areas such as building services, codes, architecture, construction, estimation, and facility management. From 1999 to 2003, the IFC standard had three major releases: Release 2.0 in 1999, 2x in 2000, and 2x Edition 2 (i.e., R2x2) in 2003. Besides these major releases, there were some other maintenance releases (e.g., IFC R1.5.1 in 1998 and R2x2 Addendum 1 in 2004), addressing minor identified issues. It is noticeable that the numbers of class definitions have been continuously increasing over the successive releases of IFC.

However, a frequently updated data model also brings chal-

¹Dept. of Civil and Environmental Engineering, Carnegie Mellon Univ.
²Dept. of Civil and Environmental Engineering, Carnegie Mellon Univ.
³Dept. of Civil and Environmental Engineering, Carnegie Mellon Univ.

Note. Discussion open until February 1, 2008. Separate discussions must be submitted for individual papers. To extend the closing date by one month, a written request must be filed with the ASCE Managing Editor. The manuscript for this paper was submitted for review and possible publication on December 21, 2005; approved on January 30, 2007. This paper is part of the Journal of Computing in Civil Engineering, Vol. 21, No. 5, September 1, 2007. ©ASCE, ISSN 0887-3801/2007/5-321-330/$25.00.
Table 1. Summary of Each IFC Release

<table>
<thead>
<tr>
<th>Year</th>
<th>Task</th>
<th>Release</th>
<th>Number of classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1998</td>
<td>Fix bug and resolve implementation issues</td>
<td>1.5.1</td>
<td>186</td>
</tr>
<tr>
<td>1999</td>
<td>Extend domain coverage</td>
<td>2.0</td>
<td>290</td>
</tr>
<tr>
<td>2000</td>
<td>New extensible platform</td>
<td>2x</td>
<td>370</td>
</tr>
<tr>
<td>2003</td>
<td>Extend coverage and increase completeness in several domains</td>
<td>2x2</td>
<td>623</td>
</tr>
<tr>
<td>2006</td>
<td>Improve IFC R2x2 and adopt a new release methodology</td>
<td>2x3</td>
<td>653</td>
</tr>
</tbody>
</table>

Table contents:

- **Table 1**: Summary of Each IFC Release
  - **Task**: Fix bug and resolve implementation issues, Extend domain coverage, New extensible platform, Extend coverage and increase completeness in several domains, Improve IFC R2x2 and adopt a new release methodology
  - **Number of classes**: 186, 290, 370, 623, 653

Challenges for civil engineers and/or architects, who both possess some domain knowledge related to the area addressed by the AEC software package and also are responsible for developing software implementations (referred to as software developers in this paper) that comply with the data models. This challenge becomes more pronounced when implementing data exchange standards because this type of data model usually is managed by another party and AEC task specific software developers may not understand these exchange standards as well as they do their private, task-specific data models that are developed for use internal to the software system. Since there are hundreds of software applications and associated underlying data models and public data exchange standards involved in the AEC domain, maintaining the software compatibility with the latest data exchange standard is a challenging task for the AEC software developers if each data model in these individual software systems must be updated when the data exchange standard is updated. Before modifying the mapping of a data model to an updated data exchange standard, software developers should know how to map the data model to both the current and new versions of the data exchange standard. To do that, they have to first understand all the changes that have been made to the new release of the data exchange standard.

A new release of a data model usually maintains the backwards compatibility by keeping a number of existing elements identical or less changed, but many new features and modifications can be inserted (Amor and Ge 2002). Therefore, it becomes time consuming and tedious to identify all changes between versions, especially when done manually (Ge 2002; Rahm and Bernstein 2001), due to the size and complexity of today’s data models used in the AEC industry. This increases the lag between the release and the deployment of the latest features of a data model. For example, in the last three years, the IFC specification has had two major updates, Releases 2x and 2x2, but many IFC-compliant commercial software applications only support Release 2.0 or even Release 1.5 (Steinmann 2004) and are far behind the latest IFC release. A manual process gets even more challenging when a model lacks a good change log document, which records major modifications made by model designers. Some data models do not have a comprehensive change log, either missing some minor changes or not specifying how a modification will affect other elements. Moreover, in certain releases of a data model, there might not be any official change log document available. Considering a data model like the IFC R2x2, where about 310 classes have been added, 60 classes were deleted, and 170 classes were modified, the lack of a change log document makes it significantly more challenging to identify all the modifications made.

A computer-aided schema matching approach can help software developers to match two releases of a data model, by enabling the identification of changes incorporated in a new release automatically. The computer-aided version matching approach should perform such a matching precisely and efficiently. “Precise” refers to the fact that it is important to be able to detect where a change has happened and what kind of a change has happened. Only identifying the existence of a change is not enough, software developers must know where to modify the existing implementations. “Efficiency” refers to the fact that the matching process should be as quick and as automated as possible, reducing human effort.

In this paper, we present a semiautomated approach that compares different releases of a data model precisely and efficiently to help software developers determine actual version differences. We created a classification of possible version differences based on the characteristics of a change, developed an approach to automatically match elements from two different versions, and validated the approach by comparing the matching results obtained with those of manual matching and prior studies in terms of precision and efficiency. The evolution of the IFC data exchange standard is used as an example in this paper, because (1) the IFC data exchange standard is supported by dozens of mainstream AEC software systems (Steinmann 2004) and is still being developed for other subdisciplines in the AEC domain; (2) it is a large-scale object-oriented data model that contains hundreds of data items and represents real world requirements; (3) it has several major releases, each of which incorporates a large variety of changes that could happen in object-oriented data models; (4) it typically does provide precise information in its change log, but also lacks a change log for some versions; and (5) it is a publicly available data model, making it easy to compare our approach with other approaches that have been developed and will be developed in the future.

Existing Approaches for Automated Version Matching

Version matching is a special case of matching between two different data schemas, where both models are variants of the same model. It has been demonstrated that a computer-aided schema matching process can help in comparing two data schemas through two fundamental automated approaches: (1) linguistic-based approaches and (2) constraint-based approaches (Rahm and Bernstein 2001). Linguistic approaches use names and/or other pieces of text to find corresponding elements by analyzing how similar they are with respect to semantics (Doan et al. 2001; Madhavan et al. 2001). Constraint-based approaches additionally consider the similarity of certain constraints, such as data types of an attribute, schema hierarchical structures, and relations between elements (Larson et al. 1989; Li and Clifton 2000; Madhavan et al. 2001; Melnik et al. 2002). These two fundamental approaches are widely applied individually or jointly, and they can potentially reduce the workload of domain experts considerably. However, approaches developed so far have the following limitations (Ge 2002): (1) most of them are tailored to a specific domain (e.g., data integration) (Rahm and Bernstein 2001); (2) some studies (e.g., Li and Clifton 2000) need to have data instances to help matching between two schemas, whereas version matching is a pure schema level matching; (3) some studies (e.g., Doan et al. 2001) still require much manual work, such as pro-
viding meta-models or identifying some candidates first for training; and (4) some studies (e.g., Melnik et al. 2002) may not handle large complicated schemas well (e.g., the IFC standard) because of complex relations between elements.

These existing approaches are not designed to handle the version matching process and do not perform version matching well (Ge 2002). In fact, computer-aided schema matching approaches can potentially address version matching better as a new version of a schema usually builds upon its previous version. Two releases of the same model usually use the same schema definition language [e.g., all IFC schemas are defined in EXPRESS ISO (1994)], share the same patterns in naming elements and organizing structures, because not only are they usually developed by the same creators, but also the creators intend to maintain backward compatibility in a new release. Hence, it is reasonable to assume that differences between releases will not be significant and a significant number of previous elements will remain identical with the same content. For example, there are about 15–25% of classes that are identical between IFC releases (Amor and Ge 2002). This feature makes version matching a much simpler problem than a general schema matching problem. Therefore, compared to matching two substantially different schemas, it is expected that computer-aided schema matching approaches will perform better in identifying version differences (Amor and Ge 2002).

This expectation is demonstrated in a recent research project (referred to as Ge-SA in this paper) that performs version matching on different subsequent versions of the IFC data exchange standard (Amor and Ge 2002; Ge 2002). Ge developed a prototype to find version differences automatically through simple comparisons of texts of two classes. The prototype takes schemas defined in the EXPRESS language as input, maps between versions, and then generates corresponding code in mapping languages, such as EXPRESS-X (EXPRESS-X 1999).

Ge defined six categories of possible changes for the class level. They are Identical, Renamed, Modified, Related, Added, and Removed (Ge 2002). The Identical category means that a source class is completely identical to a target element in all respects, whereas the Renamed category suggests that only the name is changed. Both the Modified and Related categories declare that there are differences in two versions. Whereas the Modified category suggests that both classes have the same name, the Related category does not. Added and Removed categories suggest that a new class is added, or an existing class is deleted, in the new version. Under this classification, Ge further developed some subcategories, such as the Attribute_Added and Attribute_Removed. However, those subcategories are neither reflected in the prototype itself nor tested by the test cases.

Compared to the manual matching procedure, the computer-aided schema matching approach developed by Ge works much faster (Ge 2002). For the test case comparing IFC R1.5.1 and IFC R2.0, the prototype completed the mapping in about 15 s, compared to one week of manual mapping, and about 98% of the automated mapping results were correct. On average, the prototype reports that 65% of the mapping between the two versions of the IFC could be detected automatically (Amor and Ge 2002).

The Ge-SA approach could be further improved by adding some desired features. First, the current prototype can indicate only whether or not there is a difference between two classes. It does not exactly pinpoint the location of the difference. A difference might be an explicit modification in the declaration of an element, such as changes in name, attribute declaration and constraints, or an implicit change caused by changes in referred elements. For different types of changes, software developers may adopt different ways of adjusting their implementations of the old version. Not knowing what difference exists at what location results in software developers not being able to update implementations accordingly. Hence, additional effort is still required to locate exactly where a modification occurs and change implementations accordingly. Second, Ge-SA obtains matching results by only comparing the text in the definition of two elements without considering what this text means. A text representing a different context will affect the matching result in different ways. For example, a difference in the names of two classes might be more important than a difference in the names of two Where rules that are additional constraints whose names are not important in an EXPRESS schema. Finally, when comparing text, Ge-SA requires two strings to be completely identical and thus it does not consider similar texts. This leads to a number of incorrect matching results. For example, it fails to find the relation between IfcManufactureInformation in IFC R2.0 and IfcManufacturerInformation in IFC R2x and declares IfcManufactureInformation is deleted in IFC R2x.

We built our approach upon Ge-SA and addressed the above-mentioned limitations. We first created an elaborate classification system of version differences. Then, we designed our version matching approach and developed a prototype. Finally, we used the same real world cases (e.g., releases of the IFC) as those used by Ge (Ge 2002), using a manual matching approach as a baseline, to validate our approach.

### Taxonomy of Version Differences

In reasoning about version differences, it is important to first identify which part of an element is changed, whether a change originates within that element or is caused by changes occurring in other related elements, and how it is going to affect other elements. This identification makes software developers understand the nature of a change and helps them to upgrade existing matches. Therefore, a taxonomy of version differences should be developed to address these issues. We developed two classification diagrams, representing changes at the class and at the attribute levels.

Fig. 1 shows the classification diagram for the class level changes. At the top level, four fundamental categories are defined:

1. **Identical**: An element in the new release is the same as an existing element in the existing release, including their names, attributes, and inheritances.
2. **Modified**: An existing element is changed in the new release.

![Fig. 1. Classification of version differences of the class level](image-url)
Under this category, two subcategories are further defined, based on where the change happens.

- The **Extrinsically Changed** category means an element refers to other elements which were changed. For example, an element might be referring to another class, as its parent class or declared attribute, and when there is a change in the class being referenced, the element is also changed extrinsically. The **Extrinsically Changed** category has two subcategories. One is the **Attribute Extrinsically Changed** category, where the attribute declaration is the same, but it actually refers to a modified class. The other extrinsically changed subcategory is called **Inheritance Extrinsically Changed**, which means if a parent class is changed, due to inheritance all of its child elements will be **Extrinsically Changed**. For example, in the IFC R2x2, the declaration of **IfcEquipmentElement** is the same as that in the IFC R2x, however, its parent class, **IfcElement**, is modified in terms of its attributes, so that **IfcEquipmentElement** that inherits all contents of **IfcElement** is also indirectly impacted and marked as **Inheritance Extrinsically Changed**.

- The **Intrinsically Changed** category means something changed within the description of an element, such as a change in element’s name or one of its attributes. Under this category, four subcategories are defined: **Renamed**, **Attributes Changed**, **Constraint Changed**, and **Parent Changed**. Each subcategory is a special case of intrinsic changing, indicating where the change happens. A change will be classified as one of these special types if it only contains one type of change, or as **Intrinsically Modified** category if more than one modification happens.

3. **Added**: A new element is inserted in a new release. In addition, the **Added** category has one special case, **Merged**, which means multiple old elements are combined into a single element in the new release.

4. **Removed**: An existing element is deleted in a new release. Similar to the **Merged** case in the **Added** category, the **Removed** category contains the **Decomposition** special case, which means an element is split in the new release in terms of content or functionality. The **Content Decomposition** subcategory indicates that a generic class in the prior version was replaced by several more specific classes, and the **Functionality Decomposition** subcategory indicates that multiple classes in the new release are required to perform a task done by a single class in the prior version. For example, the IFC R2x uses a single class **IfcPump** to represent pump device, whereas the IFC R2x2 uses two classes instead to implement the same function, **IfcPumpType**, which contains common properties for a specific type of pump and **IfcFlowMovingDevice**, which represents an occurrence of a piece of equipment, including a pump.

At the attribute level, a version difference classification is organized in a similar way (see Fig. 2):

1. **Identical**: An attribute is not changed in the new release.
2. **Modified**: An attribute is changed in the new release. In this category, there are three subcategories: **Renamed; Type Changed** (e.g., referring to different type); and **Constraint Changed** (e.g., an attribute is not Optional anymore). The **Type Changed** category has three special cases: **Refer to Different Type** category indicates that the attribute refers to a totally different class; the **Refered Type Modified** category indicates that the attribute refers to a modified class, which will cause the class owning that attribute to be classified as **Attribute Extrinsically Changed**; and the **Quantity Changed**

![Fig. 2. Classification of version differences of the attribute level](image)

category indicates that the capacity of a collection attribute (e.g., List) is changed in the new release.

3. **Added**: A new attribute is inserted in the new release.
4. **Removed**: An existing attribute is deleted in the new release.
5. **Moved**: The order of an existing attribute within the associated class is changed. There are two special cases under the **Moved** category. **Pushed Up to SuperType** means that attribute is moved to a parent class (e.g., **IfcOpeningElement** has one attribute **VoidElement** that is moved to its parent class when upgraded from IFC R2x to R2x2), whereas **Pushed Down to SubType** means the attribute is moved to one of its child classes (e.g., **IfcProcess** in IFC R2x2 has one attribute **Productivity** removed and that is re-introduced in its subclasses). It is notable that only a few data models are order sensitive. For example, the IFC specification is not order sensitive so that changing the order of the attributes in a class definition does not require a class to be marked as modified, so long as all the attributes are still within the same class. These elaborate classifications are able to identify typical kinds of changes occurring in an object-oriented schema, and describe changes in detail. Based on these classifications, we developed a prototype to detect version differences for a schema (e.g., the IFC) and validated the usefulness of the classification in detecting differences in various IFC releases.

**Version Matching Approach**

We developed a semiautomated version matching approach, referred to as VMA in this paper, to detect version differences based on the classifications discussed in the previous section, and created a prototype to validate the approach. The objective of this approach is to pinpoint the version differences between two versions of a data model precisely, such that the identified differences can be reasoned with to help software developers update existing implementations of the model to the new version. A good version matching approach should have the following features:

1. The approach should provide a framework that detects changes between two releases of any schema regardless of the language in which it is written. We targeted the development of a generic and extensible framework that can process multiple definition languages (e.g., EXPRESS and XML) and will incorporate new matching algorithms, as necessary. This increases the generality of the approach.

2. The approach should incorporate multiple matching algorithms and allow users to adjust different parameters used in these algorithms. There is no single algorithm that can process all variations of matching situations, therefore by integrating multiple algorithms, it is possible to avoid limitations of a specific algorithm.
The approach should be as automatic as possible to minimize required human effort. For example, when encountering a minor typographic error in the schema definition file, the approach should be able to correct it, without stopping to ask users for instructions.

Fig. 3 shows the major components of VMA. The approach takes two different releases of a schema (in the IFC case, both are EXPRESS-based files) as input and generates a series of matching pairs that link source elements (i.e., classes in the previous version) and target elements (i.e., classes in the new version). The approach works according to the following steps:

**Step 1.** Analyze the schema files of data models, which are usually represented within flat text-based files. A parser will be used to read these schema files and describe their physical structures through an abstract structure (e.g., a tree), upon which the matching algorithms actually work. Each node in this abstract structure either represents a real text label in the schema file (e.g., the name of a class) or is a supporting element inserted by the parser to define the relationship between two other nodes (e.g., a node indicating that the next node is the name of a class). Using an abstract structure significantly reduces the complexity of a matching approach (Rahm and Bernstein 2001), because it only needs to process one uniform format, instead of dealing with multiple formats used by different schemas. There are a number of parsers available. In the prototype presented in this paper, we used ANTLR, a program to generate such a parser (Parr 2004), to create a parser for data schemas, such as IFC that uses EXPRESS as its definition language. ANTLR generates a parser to analyze files defined in EXPRESS. Fig. 4 illustrates a parsed result of a class definition.

**Step 2.** The abstract structure of a schema is transformed into an object-oriented model, representing the underlying classes, attributes, and their relations in the schema file. This allows the matching procedure to easily understand which exact part (e.g., Attribute) is being worked on. These objects work as an intermediate layer between the underlying abstract structure and the matching procedure, so the matching procedure can ignore those supporting nodes that are inserted by the parser in the abstract structure. As a result, the generality of the approach is also improved because alternative parsers for generating abstract structures may be used for different cases. For the prototype presented in this paper, we used an open source project (SourceForge 2004) which can convert the ANTLR abstract structure to such objects, defining each element in the schema file.

**Step 3.** Each subcomponent of a class or an attribute is compared by one or more of the matching algorithms. For example, when comparing two classes, VMA compares their names, attributes, constraints, and inheritances (e.g., whether a class is abstract). The matching is mainly performed through linguistic-based algorithms. Besides directly comparing two texts as Ge-SA does (Ge 2002), VMA can use alternative algorithms:

1. **Edit distance** (Levenshtein 1965): This algorithm detects the number of operations needed to make two strings identical and identifies the similarity of these two strings. The smaller the number of operations, the more similar are the two texts. For example, to make IfcManufacturedInformation in IFC R2.0 and IfcManufacturerInformation in IFC R2x identical, an insert operation is required to add a letter “r” in the latter string so that the edit distance between these two texts is one.

2. **Substring:** It checks whether one string is a part of the other one (e.g., Element is a part of BuildingElement).

3. **Abbreviation:** This algorithm tries to generate an abbreviation of a phase to compare. For example, the term RotationPerMinute will be converted to RPM.

4. **Dictionary** [e.g., WordNet (Fellbaum 1998)]: A dictionary can check whether two terms have similar meanings.

VMA marks a pair of items identical if one of the above-mentioned algorithms finds the pair identical.

**Step 4.** For each class or attribute, a classifier module will collect all related matching results from its subcomponents (e.g., results of comparing its name, attributes, and constraints) and then the element is classified as being one of the version differences discussed in the above section. A source class is compared to all target classes and each comparison is assigned a type of difference. For example, if a pair of classes has different names but all attributes, constraints, and inheritances are identical, this pair of classes is declared as Renamed. Each kind of version difference has an associated matching score, which is defined by users, indicating how similar the two classes are. All target can-
didates are sorted so that the one with the highest matching score becomes the first suggestion to the user.

**Step 5.** The final matching result will be displayed in a readable report and inspected by users. The report clearly indicates the details of a match between a target object and a source object and whether there is any change. For example, if a class has a change in an attribute, the report will present whether the change comes from its name, its reference declaration, or the referred object itself. If necessary, the user can modify the matching results manually, overriding results obtained automatically. An example report shown in Table 2 illustrates that the IfcProduct Entity had its attributes changed so that IfcElement, one of its child classes, was marked as **SubType Modified** although its own declaration did not change. If necessary, the user can modify the matching results manually, overriding results obtained automatically. All the results could be saved in an external file, which will be read by other applications to reuse these matching results.

Compared to the prior study done on version matching in the AEC domain (Amor and Ge 2002), VMA offers several improvements:

**Improvement 1.** Instead of simply comparing the entire abstract structure generated by the parser, the approach works on the object-oriented layer. The parser will insert a number of supporting nodes, describing other nodes (e.g., in Fig. 4, the first node ENTITY DECL is inserted to identify that following nodes describe a class definition). In the object-oriented layer, these descriptions are converted into direct and meaningful connections (e.g., a class owns several attributes). An interesting example is that there are some special sections in the schema file using the schema’s name (e.g., “IFC_R2.0” in the IFC R2.0 case) as a scope qualifier. The text itself means nothing, but only refers to the schema file itself. When comparing such a pair of labels, one referring to the source schema and one referring to a target schema, an algorithm that simply compares texts will certainly identify two elements as different (e.g., IFC_R20.IfProduct and IFC_R2x.IfProduct) even though both elements might be exactly the same, besides the names of the schemas. However, this linguistic difference hides the fact that these texts are only references to the schema names. Our approach is able to recognize situations like this schema name reference difference and compare the underlying objects presented by each node in the abstract structure.

**Improvement 2.** VMA also tries to determine how an item that is being compared will affect other elements’ matching results. Instead of a simple Equal/Not Equal choice used by Ge-SA, each type of change is assigned a priority factor by users to define how significantly it will affect other elements when this change happens. For example, in the IFC case, if a source class has two possible matching candidates, one of which has an attribute modified, while the other only has a Where rule changed but all attributes are identical, either will cause this class to be marked as **Modified**. As a **Where rule** behaves as an additional constraint, users may consider a change in it not as important as a change in an attribute of a class. Therefore, the source class should match to the candidate with the **Where rule** changed as their attribute definitions are identical. Certainly, defining these priority factors is highly experiential, depending on the importance users assign to an element (e.g., the Attribute part usually is more important than the **Where rule** part).

**Improvement 3.** User correction is an essential part of schema matching. If the suggested matching candidate is incorrect, the user is asked to find alternative candidates. The Ge-SA can only output a candidate without identifying where the change happens. On the contrary, VMA can provide more than one matching candidate and assign priorities according to how many changes there are and what kind of changes they are. This gives users more choices and suggestions, saving time on rescanning the schema files to find another candidate manually.

**Improvement 4.** The approach is more comprehensive than the Ge-SA (Ge 2002). Although version matching is relatively easier than generic schema matching problem, a single algorithm is not enough to handle every possible case. Therefore, VMA does not depend on a single comparison algorithm, and instead incorporates multiple algorithms (e.g., Edit Distance, SubString, and Stemming). For example, the IfcManufactureInformation case could be handled by edit distance. In addition, the generality is also improved because our approach is not coupled to an abstract graph generated by a specific parser, making it possible to apply other types of parsers to process different test cases which may use another type of definition language.

**Validation**

We applied the developed prototype of the version matching approach described in the previous section on real world test cases, such as recent releases of the IFC data exchange standard and CIS/2 schemas to validate its effectiveness. The IFC schemas are large-scale, complicated, and cover most of types of changes.

---

**Table 2.** Example of VMA’s Matching Results

<table>
<thead>
<tr>
<th>IFC R1.5.1 object</th>
<th>IFC R2.0 object</th>
<th>Object type</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>IfcElement</td>
<td>IfcElement</td>
<td>Class</td>
<td>Parent class changed (see IfcProduct)</td>
</tr>
<tr>
<td>ConnectedTo</td>
<td>ConnectedTo</td>
<td>Inverse Attribute</td>
<td>Identical</td>
</tr>
<tr>
<td>ConnectedFrom</td>
<td>ConnectedFrom</td>
<td>Inverse Attribute</td>
<td>Identical</td>
</tr>
<tr>
<td>IsAssemblyThrough</td>
<td>IsAssemblyThrough</td>
<td>Inverse Attribute</td>
<td>Identical</td>
</tr>
<tr>
<td>PartOfAssembly</td>
<td>PartOfAssembly</td>
<td>Inverse Attribute</td>
<td>Identical</td>
</tr>
<tr>
<td>WR41</td>
<td>WR41</td>
<td>Where Rule</td>
<td>Identical</td>
</tr>
<tr>
<td>IfcProduct</td>
<td>IfcProduct</td>
<td>Class</td>
<td>Multiple changes</td>
</tr>
<tr>
<td>LocalPlacement</td>
<td>LocalPlacement</td>
<td>Attribute</td>
<td>Refer to a modified class</td>
</tr>
<tr>
<td>ProductShape</td>
<td></td>
<td>Attribute</td>
<td>Deleted</td>
</tr>
<tr>
<td>ProductCost</td>
<td></td>
<td>Attribute</td>
<td>Deleted</td>
</tr>
<tr>
<td>ProcessedInProcessed</td>
<td></td>
<td>Attribute</td>
<td>New attribute</td>
</tr>
<tr>
<td>Representations</td>
<td></td>
<td></td>
<td>Deleted</td>
</tr>
</tbody>
</table>
defined in our taxonomy (See Table 3). Two test cases to be discussed here include identification of version differences between: (1) IFC R1.5.1 and IFC R2.0; and (2) IFC R2x and IFC R2x2. To compare our matching results with the results of Ge-SA, we reorganized them to represent them in the same categories as those of Ge-SA. As a result, our Modified category was reorganized into three subcategories. The Renamed subcategory remains as is. All other Modified categories defined in Fig. 1 are divided into either the Modified subcategory, if the name is identical but content is changed, or the Related subcategory, if the name is changed but there are some identical contents, such as attributes.

Table 4 compares the matching results of all classes in the test case comparing IFC R1.5.1 to IFC R2.0. The IFC R1.5.1 contains 186 classes and about 650 associated attributes. The first three rows are matching results generated by VMA, Ge-SA (Ge 2002), and manual matching, respectively. Given that there are 186 classes in R1.5.1 and 320 in R2.0, the differences between these three results for each category are relatively small. Both the VMA and Ge-SA achieve similar results to the manual process. However, in terms of performance, whereas the manual results done by Ge (2002) took 1 week to produce, both Ge-SA and VMA use only a few seconds to complete.

In terms of accuracy, Ge’s automated approach achieves 98% accuracy in this case (Ge 2002). Although it is very close to the manual result, which was performed by Ge et al. (Ge 2002), there are some differences hidden behind these numbers. The most important one is that VMA exactly locates where a change happens, whereas Ge-SA only marks there is a change. In addition, the numbers also have minor differences. First, VMA has one more entry in the Identical category and one less entry in the Modified category. This is caused by a class, IfcBoxedHalfSpace, which VMA classifies as Identical, whereas Ge does not. IfcBoxedHalfSpace has one attribute that contains a scope qualifier, which refers to the schema file name (i.e., schema file names such as IFC_R15 and IFC_R20), causing the Ge-SA to improperly mark it as Modified, because two file names are certainly different as they refer to two different versions of the model. VMA recognizes the real cause of this modification and identifies the two elements as Identical as they should be. Another notable difference occurs in the Related category, where Ge-SA finds seven items and VMA only identifies two. Ge acknowledges that all of these seven matches are incorrect and the only correct Related item should be IfcMaterialComposite (Ge 2002). Our approach finds this item, with another Related item, IfcRegisteredApplication, changed to IfcApplication according to the IFC document (IAI 2003a). Therefore, our approach identifies an item that was missed in the manual matching performed by Ge and achieves a more accurate matching result.

The last row in Table 4 is generated from a change log document created for these two releases of the IFC standard (VTT 2002). Compared to the other three results, the differences are notable. The reason for more Identical classes being classified is that the change-log only accounts for changes occurring inside the definition of a class, which means it does not acknowledge extrinsic changes (e.g., where a parent is modified) as modifications and ignores changes in the Where Rule and Inverse sections of the definition of classes. For example, since the IfcRoot class, parent of most IFC classes, is modified in the IFC R2.0, all of its subclasses are classified as Modified, both in our approach and in Ge-SA, whereas the change log only marks the IfcRoot itself as changed. Meanwhile, the change log also misses some additional modifications, such as forgetting to mark IfcProjectMaterialRegistry and IfcRelCostsProducts as Deleted.

Table 5 compares the results for all classes in the version matching of IFC R2x to IFC R2x2 case. There is no official change log document for this case and when the manual matching was performed during this research, it took about 2 weeks to complete the identification of version differences, compared to less than 10 s for VMA. As in the prior case, VMA achieves similar results to the manual results.

However, the differences between VMA and Ge-SA become larger in this case study, and results from VMA are much closer to the manual results. We believe that two major reasons have con-

### Table 3. Validated Version Differences by the IFC Test Cases

<table>
<thead>
<tr>
<th>Type of version difference</th>
<th>Tested by the IFC cases</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identical</td>
<td>Yes</td>
</tr>
<tr>
<td>Modified</td>
<td>Yes</td>
</tr>
<tr>
<td>Extrinsically Modified</td>
<td>No</td>
</tr>
<tr>
<td>Inheritance Extrinsically</td>
<td>No</td>
</tr>
<tr>
<td>Modified</td>
<td>Yes</td>
</tr>
<tr>
<td>Attribute Extrinsically</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Table 4. Matching Results for Classes of IFC R1.5.1-IFC R2.0

<table>
<thead>
<tr>
<th>Approach</th>
<th>Added in IFC R2.0</th>
<th>Removed from IFC R1.5.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>VMA</td>
<td>145</td>
<td>118</td>
</tr>
<tr>
<td>Ge-SA</td>
<td>146</td>
<td>113</td>
</tr>
<tr>
<td>Manual matching results</td>
<td>145</td>
<td>119</td>
</tr>
<tr>
<td>Official change log</td>
<td>77</td>
<td>115</td>
</tr>
</tbody>
</table>

JOURNAL OF COMPUTING IN CIVIL ENGINEERING © ASCE / SEPTEMBER/OCTOBER 2007 / 327
tributed to these differences. First, VMA does not mark a class as Modified if a change happens in its subclass level, which Ge-SA does because it compares the entire abstract structure, including both its parent class and all of its subclasses. As IFC R2x2 has an additional 300 new classes, a large number of IFC R2x classes have new or changed subclasses and are recognized as Modified in the Ge-SA approach. From the object-oriented perspective, change of a parent class affects all of its subclasses because of inheritance, but the reverse does not. Therefore, other applications that try to use this version matching result may be confused when a class is identified as Modified without any change in its own and/or its parent’s content. Second, to determine if two classes have a weak connection (i.e., Related category), Ge-SA defines that no matter how many attributes a class has, if two classes have two or more pairs of Identical attributes, these two classes are Related. The changes between the IFC R2x and R2x2 classes are much larger than the ones between IFC R1.5.1 and the IFC R2.0 case. This increases the opportunity that a pair of classes without any connection has two similar attributes, and then results in more classes being classified in the Related category. Instead, VMA uses a relative threshold (e.g., 50% of attributes should be Identical) and this threshold could be tuned to achieve better results.

Table 5 only shows the differences among total numbers of categories. What it cannot display is the matching accuracy hidden behind the numbers. That is, whether an individual class is classified correctly. The IFC R2x has 370 classes, which have 965 explicitly declared attributes, 200 Where rule attributes, and 200 Inverse attributes. Among them, 15 classes are misclassified into wrong categories by VMA, and another 12 attributes are mismatched whereas their associated classes are correctly classified. As a result, the accuracy of matching results is close to 97% at the class level, which is significantly improved from Ge-SA’s accuracy (i.e., less than 40%), and 99% at the attribute level whereas Ge-SA only output limited matching information (i.e., fully matched attribute pairs) at the attribute level. VMA can even find a few pairs of related classes that are hard to discover by the manual matching approach because the change is deeply hidden in the schema. An example is that VMA identifies that IfcConstraintUsage in the IFC R2x relates to IfcRelAssociatesConstraint in the IFC R2x2, as their attributes not only have similar names, but also refer to similar types, although they have different names and different parent classes.

Another test was performed on the releases of CIS/2 schema (i.e., Versions 5 and 6). The size of CIS/2 schema is close to that of IFC, but the version differences that occurred in the CIS/2 schema were not as much as IFC’s because existing CIS/2 classes are rarely modified and most of the changes are adding new classes. Therefore, VMA can precisely identify these changes with a higher accuracy than Ge-SA approach.

Our test cases show that VMA could generate matches comparable to manual results in terms of accuracy. In addition, these comparable results are created in a few seconds, compared to weeks of effort by a manual matching approach. It took users 1 or 2 days to review the matching results of the IFC test cases. Considering the large number of data models used in the AEC industry, the VMA can potentially save significant time when matching two versions of the same data model, compared to a manual process which takes weeks of time for each version update.

Limitations of the Approach

Although the outputs of VMA discussed in the section entitled “Validation” are comparable to that of the manual matching process, several known issues impede the current approach in achieving higher accuracies. We closely investigated the data items that our approach could not match correctly and identified why there were some inaccuracies associated with those items.

One of the main cases, in which VMA does not perform well, occurs when there is a major change in the names of the corresponding classes. Usually, the name of a class is composed of several words appended to each other. In certain cases, the order of the names appeared changed and the forms of words were also changed. For example, the IfcPointLightSource in the IFC R2x schema is changed to IfcLightSourcePositional in the IFC R2x2 schema. If such a pair of classes have similar attributes, it is still possible to identify their relations; otherwise, it is hard to relate them due to limited evidence of similarity. In the IFC R2x-IFC R2x2 case, among all of the fifteen misclassified classes, there are four instances of such cases that class names are significantly changed and at the same time do not have enough pairs of similar attributes either.

Another limitation is that VMA currently lacks a means of determining Decomposition and Merged situations, although they are defined in the classification diagrams. The Decomposition and Merged changes involve 1:N and N:1 matching. These situations provide real challenges for most schema matching approaches.

Table 5. Matching Results for Classes of IFC R2x-IFC R2x2

<table>
<thead>
<tr>
<th>Approach</th>
<th>Identical</th>
<th>Renamed</th>
<th>Modified</th>
<th>Related</th>
<th>Added in R2x2</th>
<th>Removed from R2x</th>
</tr>
</thead>
<tbody>
<tr>
<td>VMA</td>
<td>136</td>
<td>0</td>
<td>157</td>
<td>8</td>
<td>322</td>
<td>69</td>
</tr>
<tr>
<td>Ge-SA</td>
<td>51</td>
<td>0</td>
<td>241</td>
<td>53</td>
<td>278</td>
<td>25</td>
</tr>
<tr>
<td>Manual matching done during the study</td>
<td>133</td>
<td>0</td>
<td>161</td>
<td>12</td>
<td>317</td>
<td>64</td>
</tr>
</tbody>
</table>

Fig. 5. Example of 1:N matching
It is hard to determine which target attribute should select for attribute ThermalLoadSource.

<table>
<thead>
<tr>
<th>Case</th>
<th>Source class</th>
<th>Target class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duplicated Where rule declaration in IfcTable in IFC R2.0</td>
<td>IFC R1.5 Entity IfcTable; ... Where WR2: SIZEOF(QRY(Temp &lt;= Rows HINDEX(Temp.RowCells)) =0; END ENTITY</td>
<td>IFC R2.0 Entity IfcTable; ... Where WR1: SIZEOF(QRY(Temp &lt;= Rows HINDEX(Temp.RowCells)) =0; END ENTITY</td>
</tr>
</tbody>
</table>

In Table 6, examples of schema that cause difficulties in matching are shown. The number of total matching errors, e.g., 15 classes and 12 attributes misclassified in the case of IFC R2x and IFC R2x2, in the two test cases reported, fixing these faults may reduce errors by about 10%.

The third issue is that errors exist in schema files that are introduced by mistake. As schemas are becoming complex in terms of scale, the existence of faults (e.g., typographic errors, inconsistencies, duplicate items, etc.) becomes inevitable, especially when a schema is integrated from the efforts of several workgroups. Table 6 gives an example in the IFC R2.0 schema, where the IfcTable class contains two additional limitations that are 100% identical except for their names (i.e., WR1 and WR2). These faults do not frequently occur in the IFC schema, with only two or three instances in one version, but they affect final matching accuracy of the matched classes. Given the low number of total matching errors (e.g., 15 classes and 12 attributes misclassified in the case of IFC R2x and IFC R2x2) in the two test cases reported, fixing these faults may reduce errors by about 10%.

The final issue involves how a schema designer creates a new schema and incorporates new features. It is hard for a designer to maintain complete backward-compatibility while adding new items. Occasionally, designers may create some confusing designs by mistake. Table 6 shows an example of the class IfcTable. In the IFC R2x, this class has an attribute, named ThermalLoadSource of type IfcThermalLoadTypeEnum and in the IFC R2x2 schema, it has two attributes, ThermalLoadSource of type IfcThermalLoadSourceEnum and ThermalLoadType of type IfcThermalLoadTypeEnum. In such cases, VMA has difficulty in determining which one to pick, with the same name or the pair with the same type and occasionally results in the approach picking the improper candidates. In the above-presented result table, VMA picked the pair with the same type, but the actual results would be that with the same name.

### Conclusion and Future Work

The AEC industry deploys hundreds of data models. A model that is being frequently updated makes it difficult for engineers who develop implementation software to follow its changes. If done manually, the determination of specific locations, where a schema has changed, is a time-consuming and tedious process. Automating the matching of two releases of the same schema is possible, as has been demonstrated by prior studies, such as the Ge-SA (Ge 2002). In this paper, we discussed an approach that addresses some of the limitations of version matching approaches developed in prior studies. We elaborated the classifications of version differences, designed an approach to detect the version differences of a schema efficiently, and tested it with several consecutive releases of the IFC data exchange standard that is widely accepted by the civil engineering software systems. This approach (called VMA) can not only significantly reduce human workload on model matching, but can also exactly point out where a change has happened and how it will affect other classes. Compared to the Ge-SA, VMA gives software developers the possibility to directly apply the matching results to upgrade existing implementations of the data model. This can potentially provide significant benefit to engineers who develop software implementations of data models to upgrade the existing implementations to apply the latest development of a schema.

Using VMA version matching, we are currently working on an approach to automatically generate correspondence between two distinct models. Given an existing matching between a source schema and a target schema, when a new version of target schema is released, the approach can possibly update the existing matching, based on the version differences detected by VMA.
Acknowledgments

This research is based on work supported by the National Institute for Standards and Technology under Grant Nos. 60NANB2D0158 and 70NANB3D1114. The writers are also grateful for the support from Professor Robert W. Amor.

References


