Theme:

Title: **Modeling and Implementation of Smart AEC Objects: An IFC Perspective**

Author(s): Mahmoud R. Halfawy and Thomas Froese

Institution(s): University of British Columbia, Canada

E-mail(s): halfawy@civil.ubc.ca, tfroese@civil.ubc.ca

Abstract: *Smart AEC objects are an evolutionary step that builds upon past research and experience in AEC product modeling, geometric modeling, intelligent CAD systems, and knowledge-based design methods. Smart objects combine the capability to represent various aspects of project information required to support multi-disciplinary views of the objects, and the capability to encapsulate "intelligence" and "knowledge" by representing behavioral aspects, design constraints, and life cycle data management features into the objects. An example implementation of smart objects to support integrated design of falsework systems is discussed. The paper also discusses the requirements for extending existing standard industry-wide data models, specifically the Industry Foundation Classes (IFC), to support the modeling of smart AEC objects.*

Keywords: *Product models, smart objects, intelligent design systems, IFC*

Introduction

AEC project information typically flows from the design phase to the construction to the facility management phase, with very costly and time-consuming feedback loops in the form of change orders and rework during the construction phase, or excessive maintenance work during the facility management phase. Due to the highly interdependent and multi-disciplinary nature of AEC objects, this uni-directional style of information flow has often resulted in project cost and time overruns, reduced quality and maintainability, loss of design intent, and the inability to efficiently access and exchange objects information. AEC objects typically involve large, complex, and dynamic information structures. Objects' data span several domains (e.g. design, specification, cost, schedule, etc.) and involve the structural, functional, and behavioral characteristics of the object. A typical facility is comprised of a large number of objects with complex dependencies and relationships. Also, different project disciplines usually have different views of the same object, and each view defines its own set of object attributes and methods.

AEC systems have evolved from being purely analytical to drafting tools to design tools. Although the systems are becoming more efficient and sophisticated, their focus was primarily limited to the detailed final stages of the design. Most systems still exhibit some fundamental limitations: representing only the geometric aspect of the AEC objects; objects do not represent functional or behavioral attributes and therefore have no intelligence; systems are generally used after the design is substantially complete, mainly to output the design in the form of drawings; design changes have to be manually propagated to all geometric representation of the affected objects which, besides the time and cost involved, could result in inconsistent project documents.

Attempts have been made to overcome some of these limitations through the use of AI and knowledge-based techniques to add more intelligence to the objects to support early conceptual design stages. During the last two decades, numerous schemes for systematizing and representing design knowledge, cognitive computational models, design theories, and frameworks have been proposed (e.g. Gero, 2000). Despite the significant body of knowledge currently available in this area, these models rarely produced practical software solutions to the AEC industry. Most of the developed systems were primarily experimental research prototypes that could not gain the interest of the industry.

Parallel to these efforts, another thread of research was focusing on the data modeling of AEC objects. Although this area started as part of the AI approach, it evolved into a separate research thread within the AEC IT community that primarily focused on the use of standard data models to enable systems integration and interoperability. Several data models for AEC objects were proposed. Examples include GARM, PISA, ATLAS, COMBINE, RATAS, OPIS, ICON, COMBI, and VEGA, just to name a few.

(Eastman, 99) provided an excellent review of many of these models, which have, undoubtedly, contributed significantly to our ability to represent AEC objects data. Many AEC models can now represent objects, their properties, and their inter-relationships in a comprehensive and accurate manner. Many efforts have been underway to standardize the representation of objects data to enable the exchange of project information in a standard and neutral format.

This paper presents an evolutionary step that aims to build on past research and experience in AEC product modeling, geometric modeling and intelligent CAD systems, and knowledge-based design techniques, in order to support the modeling and implementation of "smart" AEC objects. Smart objects combine the capability to represent various aspects of project information required to support multi-disciplinary views of the objects, and the capability to encapsulate "intelligence" and "knowledge" by representing objects' behavioral aspects, design constraints, and life cycle data management features. We view this approach as a major enabler to support the development of next-generation interoperable, integrated, and intelligent AEC systems.

State of the Art of AEC Product Modeling

Developing standard data models has been a major thrust for academic research during the past decade. Many of the object models have reached a high level of maturity and sophistication in supporting a wide range of project aspects. Most notably, the Industry Foundation Classes (IFC), developed by the Industry Alliance for Interoperability (IAI) (IAI, 2002), now represents a standard AEC/FM data model that is widely accepted and supported by the industry. The IFC model defines an integrated schema that represents the structure and organization of project data in the form of a class hierarchy of AEC objects. The schema defines the main data objects, their characteristics, and their inter-relationships. The IFC class hierarchy covers the core project information such as building elements, the geometry and material properties of building products, project costs, schedules, and organizations. Instances of the IFC are initialized, linked, and assembled by application software to create an object model of the building project. Generally, the information from many types of computer applications can be mapped into IFC data files. In this way, IFC provides a standard data model and a neutral file format that enables applications to efficiently share and exchange project information.

The IFC model is the culmination of over a decade of research and development. The model has undergone four major releases, and many commercial software tools have already implemented IFC file exchange capabilities. Using the IFC project data model could significantly improve the availability, and consistency of project information, and would serve to integrate the multi-disciplinary aspects of the projects and facilitate the exchange of project information between function-specific software tools. As a result, this would minimize the need for human intervention to re-interpret and re-format the data to marshal it between various tools, and thus eliminating the possibility of errors during data transformation.

In the simplest form of interoperability, the project model is communicated from one application to another in a data file (e.g. using ISO 10303 Part 21 format). Upon receipt of the data file, the software will re-create the project model for further processing. As an example of the current capabilities of IFC-based file exchange, the following scenario has been implemented using tools developed by the Building Lifecycle Interoperable Software group (BLIS, 2002).

- One tool is used to define the basic rooms and spaces of a building, including the names, areas, and other basic requirements. The resulting preliminary space plan is exported to an IFC data file.
- The IFC file is read into a two-dimensional technical drawing tool. In this tool, previously identified rooms are arranged into an overall floor plan, and various design details such as windows, doors, plumbing and mechanical systems, are added. The resulting design is then exported as an IFC file.
- The IFC file is opened by an energy analysis tool. Although the information had previously been constructed in a 2D drawing package, all of the elements have height and elevation properties, and can form full 3-D models in the CAD system. This tool performs energy simulations, and allows design revisions to the HVAC components, with the results again exported to an IFC file.
- The IFC file is opened in a tool that generates a 3-D virtual reality view of the project, allowing the user to rotate, zoom in and out, and walk-through the building. This tool then runs a series of design

checks; rules which look for specific code conformance issues. Items that fail to pass the design checks, such as a room with insufficient fire egress provisions, will be highlighted in the 3-D model.

- The IFC file is opened in a tool that itemizes all of the physical components in the building, and maps their properties to an estimating database, to build up a complete cost estimate for the project.

This scenario describes the current state-of-the-art in IFC-based information integration. With the basic product modeling capabilities of IFC now reaching a high level of maturity and stability that proved to be successful in many project scenarios, we are focusing on defining and developing ways to extend the model semantics and intelligence, data exchange mechanisms, project areas, and application domains. Specifically, we are working to extend the IFC model in several main directions:

- *Adding support for smart AEC/FM objects*. Extending the IFC model to enable the encapsulation of objects intelligence and knowledge into the model. We are investigating adding new IFC classes to add support for modeling smart AEC objects that could enable representing richer semantics regarding objects behavior, management of evolutionary object data, and representation of the objects design rules, constraints, and procedures. This paper focuses on this direction.
- *Moving beyond file-based data exchange*. Current implementations of IFC-based integration rely almost exclusively on the exchange of IFC files. This simple mode of transferring data is very limited in its ability to manage a large pool of shared project information that is accessed concurrently by many users, or to enable transactional forms of data exchange between project parties and applications. The development of system architectures for distributed systems, such as IFC-based centralized object-oriented project repositories, is the next logical step.
- *Moving beyond ad-hoc transactions*. While the IFC model standardizes the information content of an information exchange transaction, it offers no guidance to the context of these transactions. It is still left up to the two parties exchanging information to come up with ad-hoc agreements about what data are being exchanged, for what business purpose, with what constraints and obligations on each participant, etc. We are pursuing the formalization and possible standardization of data exchange protocols to support IFC-based transactions in distributed and heterogeneous environments. An accompanying paper in this proceedings focuses on the process of developing transaction-based interoperability protocols.
- *Extending the IFC model to address project areas that are currently not supported.* This could be achieved either by referencing and linking with other data modeling schemas (e.g. CIMSteel for structural design), by adding new property sets (e.g. to model specifications), or by introducing new IFC entities (e.g. facilities management classes).
- *Extending the IFC application domains*. The IFC model specifically addresses building construction. Yet much of the content of the model is fairly generic and could be applied to other segments of the industry (e.g. bridges).

The next section discusses the fundamental characteristics of smart AEC objects. Then, the implementation of smart AEC objects to support integrated design of falsework systems is presented. Finally, the requirements to extend the IFC model to support the modeling of smart object are outlined.

Characterization of Smart AEC Objects

Smart AEC objects are semantically rich, product-centric data models that include an AEC object or object assemblies that represent not only the data attributes of these objects, but also encapsulate object-related behavior and intelligence in the form of behavioral attributes, objects inter-relationships, and design rules and constraints. Smart objects are represented by a set of parameters (i.e. parametric attributes), methods, and a set of rules and/or procedures. Smart objects are particularly suitable to support configuration design problems where objects could be fully described using a set of configuration parameters and rules that specify configuration constraints on the object and on sets of dependent objects can be defined. Six basic characteristics for smart AEC objects have been defined:

First, a distinguishing characteristic of smart objects is their behavioral intelligence. Objects implement methods to ensure the integrity of their components and validity of their structural, functional, and behavioral parameters. For example, an object may adjust the values of some of its parameters based on changes in other parameters, or an object may re-configure or re-locate some of its components in

response to changes in other related objects. Objects also implement behavioral features to ensure intelligent and realistic interaction with other objects in the system. For example, re-configuring or re-locating one object may automatically cause re-configuration or re-location of other objects.

Second, a smart object can manage its evolutionary state throughout the project life cycle. Moving from conceptual to preliminary to final design stages, smart objects can keep track of their evolution history. Two possible evolution changes can be tracked: changes in object definition (or schema) and changes in object configuration (i.e. parameter values). Changes in objects' schemas are tracked by maintaining a "version identifier" for each schema and defining methods to map between different schemas. Changes in object configuration are tracked by maintaining a list of the object "parameter set" along with a change version identifier and change information. Each object has a current "parameter set," from which previous sets (or object states) can be navigated.

Third, smart objects can be arbitrarily complex by aggregating or referencing other objects and defining the rules that describe the inter-relationships and interaction between these objects. The advantage of defining objects as compositional assemblies of more primitive objects is two-fold: higher level objects are more intuitive and easier to work with from a design viewpoint; configuration rules and design constraints can be enforced to control complex objects configuration at higher level objects and therefore, ensure the consistency and validity of the design at all design stages.

Fourth, objects parameters not only describe the geometry but also describe other non-geometric information such as material, specification, cost, construction methods, etc. Smart objects integrate project information by representing and considering the interaction of different project disciplines at the object level. That is, the object model will glue together different project views pertaining to the object and enable interoperability of function-specific software tools addressing these views. Using the same object model, different project disciplines can access data relevant to their domains. Also, integrating different project views would enable developing methods to automate mapping between these views (e.g. automatically performing quantity takeoff).

Fifth, three-dimensional geometric representation of smart objects can be constructed given the set of object parameters. Addressing configuration design problems requires accurate geometric representation of objects and their spatial relationships (e.g. for objects layout, interference detection, etc.). The use of 3D models would provide a wide range of benefits to various project processes. Project team members can more easily review and evaluate the design details from multiple perspectives to identify potential problems. This could significantly improve the communication and collaboration among project teams.

Sixth, smart object models can serve as the building blocks for knowledge-intensive design environments (Tomiyama and Mantyla, 98). Smart objects models support attaching computable or non-computable forms of objects-related knowledge. Computable knowledge can be represented declaratively in the form of production rules (e.g. configuration constraints or design rules) or procedurally in the form of computational methods (e.g. for calculating structural responses or materials quantity). Non-computable knowledge, such as unstructured documents, can also be linked to specific object parameters. Making object-related knowledge accessible through the object model could be used to automate the computation of some objects parameters and to provide an efficient method to index and access project documents. This will also allow the use of the object model to capture, collect, and represent the design knowledge and expertise in the form of object-centered design knowledge bases (Yoshioka et al, 98). As design becomes increasingly knowledge-intensive and collaborative, the need to support the capturing, representation, and use of design knowledge becomes even more critical.

Implementation of an Integrated Falsework Design System Using Smart Objects

A prototype software environment was implemented using smart objects to support the integrated design, layout, structural analysis, cost estimating, erection planning, and inventory management of falsework systems used for cast-in place concrete box girder bridges. Falsework objects can be thought of as specialized assemblies of the IFC columns and beams elements that are designed and erected to satisfy structural, schedule, and site constraints, among others. The software was implemented using the ObjectARX class library that extends the AutoCAD environment (AutoDesk, 1999). The system was developed in collaboration with a Taiwan-based falsework subcontractor (Hua Construction, Inc.)

Falsework systems are complex structures in terms of the number of geometric and functional parameters that a designer must consider in order to develop a structurally safe and economical system. Falsework systems are typically designed and erected by a specialty contractor who needs to share and exchange project information with both design and construction teams. The input to this process typically consists of the bridge design documents, site topography (including soil data), and the construction schedule. The falsework subcontractor team will then define the system design parameters along with the specification, cost estimating, and erection planning to meet the bridge design and construction schedule constraints. This process typically requires several iterations and is subject to stringent code requirements to ensure the stability and safety of the system.

Falsework smart objects are developed to support efficient design as well as the sharing and exchange of project information between various project disciplines. The objects are implemented as ARX classes. These objects included the falsework segments, towers, beams, and grids. The object model encapsulated rules to ensure the consistency and correctness of the design throughout the project stages. A falsework system is comprised of a series of straight or curved segments. Each segment contains an array of towers positioned at regular or irregular grid points. In laying out a segment, the user specifies the segment parameters which include length, width, start and end offsets, number and spacing of towers in each direction, the dimensions of each tower, and the ground elevation and top elevation at the start and end of the segment. The user can define any number of segments. The user can also modify segments' global properties (e.g. its elevation) as well as the location, height, and dimensions of individual towers. Dialogs for editing the grid dimensions and towers configuration are also implemented. The user can also manually or automatically place layers of beams longitudinally or transversely in a segment. The system has been developed to primarily support the following use cases: 1) Create, configure, and modify falsework segments; 2) Perform structural modeling and analysis; 3) Produce layout drawings; 4) Produce bill of materials and cost estimates; 5) Planning and visual simulation of the erection schedule in relation to the bridge construction schedule; and 6) Edit the object library. Figure 1 shows a sample view of the falsework system.

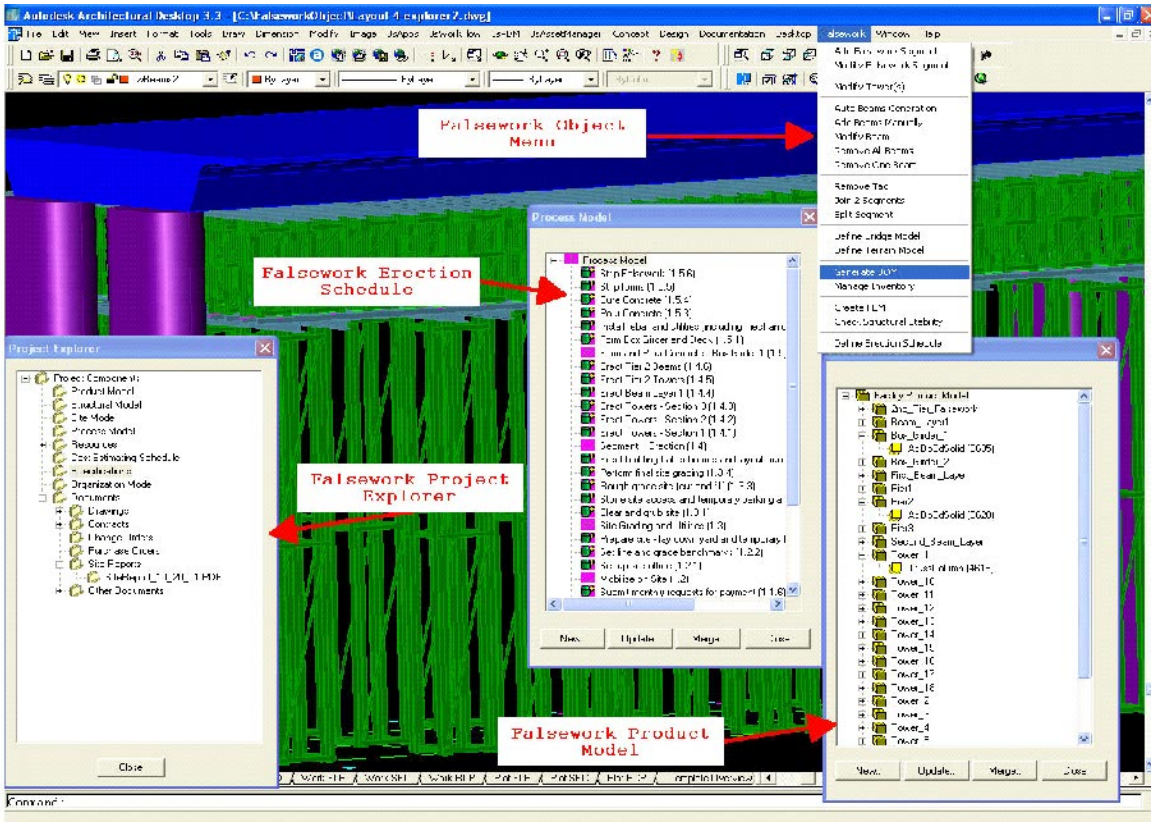

Figure 1: Different project views supported by the falsework smart objects

A number of different configurations for each object are defined in a library and a falsework designer would select the components and configurations that are most suitable to the project at hand. Designers then specify values for the pre-defined objects parameters. Based on these values, the objects determine their 3D geometric configuration and can perform several functions such as generating a bill of materials for cost estimating or a finite element model to check the stability against different loading conditions. Designers position the objects in their exact location by referencing points on the bridge structure or on the site. Objects behavior was modeled and implemented in the form of methods to control their response to user modifications to ensure that the object will remain in a consistent and valid state and that its relationships to other objects are maintained. Designers can modify different objects (e.g. tower heights) and the impact of these changes is automatically propagated by the object methods to other dependent objects (e.g. beams layers). Objects also defined methods to store, access, and manage their multi-disciplinary project information (Figure 2). By encapsulating the data relevant to different project disciplines, it would be possible to automate and support inter-related project tasks across multiple disciplines while ensuring the consistency of the project data. For example, if the objects' configuration changes, the objects can modify the structural model and re-perform structural analysis, recalculate the quantities to be used for cost estimating, or perform checks to ensure that the new changes are within the code limits.

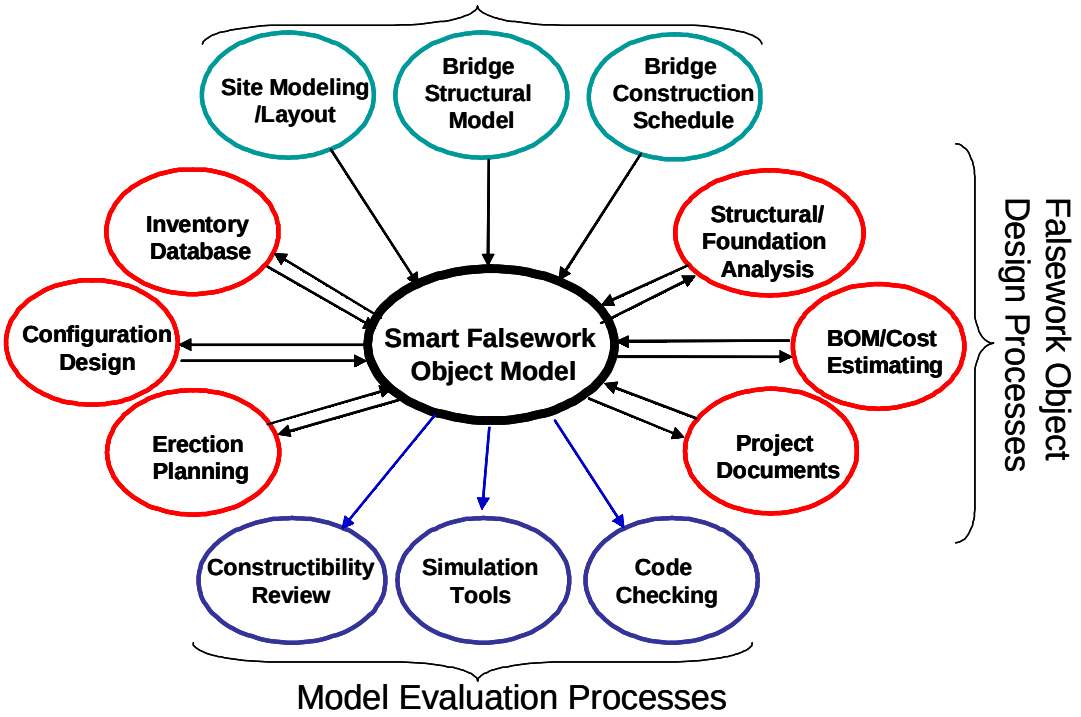

Figure 2: Integration of falsework project processes using smart objects

Requirements for Modeling Smart AEC Objects in IFCs: Beyond Product Data Modeling

The capabilities offered by smart objects cannot currently be obtained using existing data models, including the IFC model. The IFCs focus primarily on data modeling and do not address behavioral aspects of the objects. Also, traditional knowledge-based techniques do not address the modeling of the multi-disciplinary objects' data in a complete and comprehensive manner. Representing a hybrid of the two approaches, smart objects are positioned to play an integrative role that would build on the strength of both approaches to allow software tools to share and exchange semantically rich AEC object models. Below, we present a set of requirements to extend the IFC model in order to support the modeling of smart objects. These extensions would enable the addition of more semantics (i.e. behavior and knowledge) to existing IFC objects and allow the definition of application-specific custom complex objects that are composed of more primitive objects. In some places, we briefly proposed ways to satisfy these requirements following the general IFC modeling.

The IFC model defines a flexible and powerful mechanism that allows extensions to the model through the use of the IfcPropertySet entity. An IFC property set could be used to define a set of properties (IfcProperty entities or other nested IfcPropertySet entities), and can be linked to any number of IFC objects using the IfcRelAssignsProperties entity. Using this approach, we could, for example, define a property set that describes the specifications of a building element and link this set to the IfcProduct entity that represents a specific building element using an IfcRelAssignsProperties entity. Similarly, cost estimating, construction planning, or facilities management information could be linked to IFC objects.

Representing and linking objects' behavior can be supported using a similar approach. A new IfcBehaviorDef entity could serve as a container for object-related rules and procedures, both of which are supported by the EXPRESS language. IfcBehaviorDef Objects can be linked to any number of IFC objects using another entity called IfcRelAssignsBehavior. Besides describing object behavior and interaction with other objects, an IfcBehaviorDef object can be used to capture and represent structured object-related knowledge in the form of rules and procedures. Unstructured knowledge, in the form of documents, can be defined using properties in a property set linked to the object.

In a typical IfcBehaviorDef entity, rules may reference the attributes of the same object or other objects. To execute these rules, IFC toolkits would need to implement or to interface with a rule-based engine, such as CLIPS. Also, the procedures are defined by their interface (i.e. signature) and could be implemented using standard component interfaces (e.g. a standard COM interface), or using web services. Smart objects would need to access these components or web services to execute these procedures. For example, an "CalculateFalseworkBillOfMaterials" procedure may be implemented in a standard interface (e.g. ICalculateFalseworkBillOfMaterials), or as a web service, to compute the bill of material of the falsework system. Other procedures could be implemented to check some object values or to retrieve some data (e.g. form an online product repository).

Another needed extension is the capability to represent arbitrarily complex objects assemblies where an object can be composed of more primitive objects (e.g. a tower object that is composed of column and beam primitives). The IFCs enable modeling of a pre-defined set of complex objects (e.g. building, storey). A new IfcProduct-derived entity (e.g. IfcComplexObject) could be used to support this extension. This entity will define a list of references to its component objects. This would enable the modeling, managing, and accessing arbitrarily complex and hierarchical AEC objects as one unit. To define the inter-relationships among the set of component objects, an IfcBehaviorDef entity could be used to specify the "rules of interaction" between these objects. The IfcBehaviorDef object will be linked to the IfcComplexObject instance using an IfcRelAssignsBehavior object.

Automating objects' change propagation is another requirement to support behavioral and complex object modeling. Changes could occur within the same object or in dependent objects. For example, changing a tower type may require changing certain dimensions, or changing the towers height would require changing the supported beams elevation. Constraint and behavioral rules can be formulated to enable automatic propagation of these changes. For example, a rule stating that "if tower type is X then dimension Y = Z" could be used to represent the dependency between a tower type and dimensions, while a constraint stating that the beams' elevation equals the towers' top elevation could be used to propagate the second change. A third type of change that needs to be propagated concerns changing the location of an object and its impact on other dependent or connected objects. For example, moving the towers would also require moving the beams. Supporting the propagation of this type of change would require adding an entity that represents "connection" or "anchors" between different objects as well as a mechanism to detect that changes have occurred in some objects. Anchors can be thought of as another form of relationships between objects that indicates "spatial-dependencies" between objects.

Another possible extension feature concerns managing the evolutionary objects data throughout the project stages. The objects are initially defined using a simple set of parameters and evolve into a more comprehensive and detailed representation as more design iterations are performed. The IFC model defines an entity, IfcOwnerHistory, which supports tracking the agent who performed the change. However, it does not define any mechanism to track the changes themselves. Therefore, a possible IFC extension to support smart objects would be to enable managing the evolution and change of the objects' data sets. This capability is equally important for smart complex objects and for IFC products currently defined in the model. A possible approach could involve the use of property sets and a labeling scheme to identify IFC products as as-designed, as-changed, or as-built, and to track the history of changes in each

modified object. Objects labeled “as-changed” would define a version number along with change information, and a reference to the original object. After the design and construction phases conclude, all products will be labeled “as-built,” which could be transferred to the facilities management tool to create an asset inventory and prepare the FM database.

By supporting smart objects capabilities in the IFC model, systems will no longer merely exchange objects static geometric data, but more knowledge and semantics about design, including behavior, design rules, constraints, rationale, etc. Design rationale will be more explicitly defined and exchanged, and objects data will be better managed across all project stages and throughout the project life cycle.

Conclusions

Smart objects are an evolutionary step that builds on almost two decades of research and experience. This paper discussed the main characteristics of smart AEC objects and presented the requirements to extend the IFC data model to support the modeling of smart objects. Besides serving as data models that integrate multi-perspective project views and encapsulate behavioral object intelligence, smart objects also enable exchanging semantic-rich data models between different software tools. Multi-disciplinary project teams could use these rich models to share and communicate design rationale and to evaluate the impact of changes on inter-dependent objects (e.g. cost or schedule implications of a deign change).

Smart objects could potentially achieve many benefits to the design process: (1) Modeling behavioral aspects within individual objects as well as between inter-dependent objects, and integrating those aspects with the structural and configuration aspects of AEC objects; (2) Automating routine decisions and design checks that could be based on geometric or non-geometric objects parameters; (3) Automating propagation of changes and enforcing design constraints and rules to maintain the design consistency and validity; (4) Integrating design with other project activities (scheduling, estimating) and thus facilitating the information sharing across project activities; (5) Reducing the time needed to design complex objects and allow designers to focus on design issues and to perform quicker design iterations; (6) Provide feedback to designers if any constraints or requirements are violated; (7) Automatically build 3D models from parametric object description; and (8) Capturing and encapsulating design knowledge into these objects.

Acknowledgements

We gratefully acknowledge support for this work from the Natural Sciences and Engineering Research Council of Canada, Collaborative Research Opportunities Program, and from Hua Construction of Taiwan.

References

1. AutoDesk Inc., ObjectARX Developer’s Guide, (1999).
2. BLIS, Building Lifecycle Interoperable Software, Home Page at http://www.blis-project.org, (2002).
3. Eastman, C.M., Building Product Models, CRC Press, Boca Raton FL, (1999).
4. Halfawy M.R. and Froese, T., A Model-Based Approach for Implementing Integrated Project Systems, Proceedings of the Ninth International Conference on Computing in Civil and Building Engineering (ICCCBE), April 3-5 Taipei, Taiwan, (2002).
5. IAI, International Alliance for Interoperability, http://www.iai-international.org, (2002).
6. Gero, J. S. (ed.), Artificial Intelligence in Design’00, Kluwer, Dordrecht , 719pp, (2000).
7. Wix, J. and Liebich, T., Industry Foundation Classes Architecture and Development Guidelines, International Alliance for Interoperability (IAI), (1997).
8. Yoshioka, Y., Shamoto, Y., Tomiyama, T., An application of the knowledge intensive engineering framework to architectural design, In and Finger, S., Tomiyama, T., Mantyla, M., editors, Knowledge Intensive CAD-3, Workshop on Knowledge Intensive CAD-3, (1998).
9. Yoshioka, M., Sekiya, T., Tomiyama, T., Design Knowledge Collection By Modeling, Proceedings of the Tenth International IFIP WG 5.2/5.3 Conference PROLAMAT, Trento, Italy, (1998).