www.itcon.org - Journal of Information Technology in Construction - ISSN 1874-4753

THE DESIGN BRIEF: REQUIREMENTS AND COMPLIANCE

SUBMITTED: March 2016
REVISED: September 2016
PUBLISHED: November 2016 at http://www.itcon.org/2016/22
GUEST EDITORS: Dimyadi J. & Solihin W.

David Marchant, Senior Research Fellow,
UNSW Australia, Sydney, Australia;
d.marchant@unsw.edu.au

SUMMARY: *The activities of briefing (stating problems to be solved) and designing (instantiating solutions) are intimately interlinked – both parts of the same process to develop and procure a product which satisfies one or more needs. There is an on-going "conversation" between problem (requirement) definition and design proposal in which acceptable proposals progressively add to, and refine, the definition of the whole solution while at the same time potentially generating further problems to be resolved. At any point in time through this process, the developed whole solution is composed of partial solutions which form the context against which further problem statements are made. These problem statements are indicators for a future desired state of the whole solution. The design briefing process also can be seen as starting before professional designers are involved and continues after they complete their project contribution, so retaining design intentions alongside solutions is valuable for the on-going use of a designed product. With these ideas as background, this paper investigates the hypothesis that data for briefing and design can be usefully correlated within integrated building information data models. Several extensions are proposed to the IFC data schema that formalise the way requirements can be modelled. It is then shown how rule-based checking of solutions can be implemented against requirements defined in this way.*

KEYWORDS: *Briefing, Building Information Modelling (BIM), IFC, compliance checking.*

REFERENCE: *David Marchant (2016). The design brief: requirements and compliance. Journal of Information Technology in Construction (ITcon), Special issue: CIB W78 2015 Special track on Compliance Checking, Vol. 21, pg. 337-353, http://www.itcon.org/2016/22*

1. INTRODUCTION

An integral component of the design process for a product and the on-going use of that product is that it should satisfy requirements of various kinds. Requirements may come from the client who commissions the product, from the designers themselves, or from one or more regulatory authorities (for example, usage, quality, and safety standards). The products of the construction industry are built facilities: buildings, bridges, roads, railways, and other man-made structures. Many of these products are now designed and managed digitally, and there is the emerging ability to locate and interact with that digital information from a shared data repository for the use of all of the participants in a design project (BIM Industry Working Group, 2011).

Compliance checking of building information models has been addressed by many authors for more than 30 years (Gero 1982, Rosenman et al 1986, Balachandran et al 1991, Fenves et al 1995, Drogemuller et al 2000, Woodbury et al 2000, Maissa et al 2002, Ding et al 2006, Niemeijer et al 2009, Pauwels and Zhang 2015). Using available technology at their time of writing, these authors have proposed that clauses from statutory and other design codes can be reformulated as if-then rules, or constraint statements, where the variables that each rule operates on are inferred directly or indirectly from the objects, their properties, and their inter-relationships in the data model that is the representation of the product being validated.

Because, in practice, designers and authorities deal with many different product designs, a single standard for structuring the data in building information models is desirable as a basis for common understanding, interpretation, and validation. The international effort towards standardizing the representation and exchange of computer-interpretable data concerning products (man-made objects) began in 1984 with the commencement of work on the International Standards Organisation standard ISO10303, Automation systems and integration – Product data representation and exchange. ISO10303 is a huge multi-part standard that, for many industry sectors, provides a system-independent means to describe a product throughout its life cycle – including geometric as well as non-geometric data. Informally known as STEP (Standard for the Exchange of Product Model Data), it includes a formal data modelling language (EXPRESS) as well as application protocols to address particular industry domains and other resource parts. The architecture, engineering and construction (AEC) industry were involved from the beginning, but ultimately formed a parallel standardisation group (the International Alliance for Interoperability, later rebadged as buildingSMART) which produced the Industry Foundation Classes (IFC) data schema outside of the formal ISO committee setting. The IAI/buildingSMART have produced several versions of building product model specifications for the AEC industry – official releases are IFC1.5, IFC2.0, IFC2x2, IFC2x3, and currently IFC4 (2013). The IFC2x3 release was submitted to ISO, where it was accepted as ISO 16739:2013 (2013). IFC contains an extensive set of computer-interpretable entity classes and their properties for building works: spatial elements (sites, buildings, building storeys, and spaces); physical elements (walls, floors, doors, windows, furniture and equipment, services, and many more); actors (persons and organisations); processes; and, the relationships between entities. These classes have been defined using EXPRESS.

The DesignCheck tool developed by Ding and her team (including this author) uses IFC. The tool takes an IFC building model as input, extracts and recasts relevant entities into its own internal data model, then validates those entities against programmatically-defined rule sets. As a proof of concept, the encoded rule sets are: Australian Standard AS1428.1 - Design for access and mobility Part 1: General requirements for access – new building work; and, Building Code Australia New draft access code for buildings Part D – Access and egress (D3). From these, where possible, the semantic content of each statutory requirement is reformulated as a rule written in EXPRESS. The rule base also contains functions to derive higher-level semantics for use in the application of the rules – for example, to determine the ordered series of spaces and doors along a required egress route.

The following is an excerpt from one of the rules in the DesignCheck rule base. It finds all space objects in a building with the attribute “Publicly_Accessible” set to a value of “True”, and checks the value of the OverallWidth and Handicap_Accessible attributes of all the door objects that are related to each of those identified spaces to identify whether or not all spaces in the building that are designated as accessible do correctly comply:

```
REPEAT i := LOINDEX(Space_CRC) To HIINDEX(Space_CRC);
     space := Space_CRC[i];
          IF ((space.Publicly_Accessible = True) OR (space.Handicap_Accessible = True))
          THEN
              doors := Get_DoorSet(space);
              REPEAT j := LOINDEX(doors) To HIINDEX(doors);
                   door := doors[j];
                   IF (door.OverallWidth < 800)
                   THEN
                        Clause_7_2_1 := 'Non-Compliance';
                        DoorFailedSet ++ door;
                   END_IF;
              END_REPEAT;
          END_IF;
END_REPEAT;
```

The rule-based approach has been shown to work where a requirement clause (from a regulation or standards code) is amenable to coding in this way. However, requirements may be expressed purely as "vision" statements that will require value judgements by humans to determine compliance. Or, where codes define acceptability in terms of "deemed-to-comply" clauses, the onus is on the designers to demonstrate that their proposed solution satisfies the intent of that performance-based requirement. In this case, computer simulation of an acceptable percentile of possible use cases may be another way to validate conformance. For example, compliance of a design with requirements that relate to access for people with disabilities could be demonstrated by simulating movements within a space, or set of spaces, by various types of avatars (actors) having a relevant disability. Unlike rule-based techniques, simulation is not a fully automated checking methodology because (at least at present) it requires human interpretation of the solution (or solutions) demonstrated. Whether a rule-based, a simulation-based, or some other methodology is employed to validate a design model, the entities in that model need to be distinguishable in a form that the chosen method can operate on (or, in the case of human interpretation, understand syntactically and semantically). These entities represent the set of solutions to requirements at a point in time. Rules, as in the DesignCheck example above, encapsulate the requirement statement in the functional code. However, instead of conjoining requirements with the compliance method, is it possible to invert this paradigm and represent the requirements (the design brief) within the product data model (either explicitly or via external referencing)? The compliance method may then take a requirement as input, and use the syntax of the requirement to examine the solution entities in that data model to determine and report on compliance, as described later in this paper.

2. DESIGN BRIEFING

In architectural design, briefing (or programming, as it is known in the US) is concerned with defining the context, vision and client requirements for a proposed building project. Pena (1977) refers to the goal of briefing as to 'state the problem', and Wade (1977) describes the inherent inter-connection of problem definition (briefing) with solution generation (designing). Blyth and Worthington (2001, p. 3) distinguish the act, or process, of briefing, from the outputs of that process: 'briefing is the process by which options are reviewed and requirements articulated, whereas a brief is a product of that process'. Previous research that has addressed IFC in relation to briefing includes Kiviniemi's (2005) research on requirements management. Two IFC extension projects have also dealt with briefing issues: the FM-9 Portfolio and Asset Management – Performance

Requirements project (PAMPeR, 2004); and, the AR-5 Early Design project (AR-5, 2006). More recently the Building Programming Information Exchange project (BPie, 2012) has been defining common requirements for space and functional briefing.

While traditional design practice makes distinctions between stages of design (briefing, sketch design, detail design, and documentation) in practice these distinctions are much more fluid. As the design progresses, and new knowledge is developing, so the concerns of the briefing adjust and vice versa. Briefing requirements and design decisions relate to the level of detail being addressed as well as to who the intended resolver of those requirements might be. For example, briefing at the detailed room level is at a stage where implicit decisions have already been made – the departments and rooms are identified, only missing final positioning and finessing of their design detail. In effect, a brief at this stage can be understood as a partial solution to the on-going design problem. In the client's consideration, prior to this are factors such as organizational configuration, financial viability, and business processes, many of which are relevant to intentions for the final design. Where the room is the organizing entity for detailed room briefing, the organization, business unit and people are relevant organizing entities prior to considerations of space. Additionally, requirements can often be contingent (placeholders) just to get the design moving in the desired direction, but once the design is underway those requirements can be subject to more rigorous consideration and challenge, and therefore also subject to change.

In a building construction environment where risk can be high, profit margins are generally low, and communication fragmented, integration around a shared understanding of the building project can help to reduce waste caused by misunderstandings and lost information. Additionally, because the delivery times for a building project can be relatively long (years) and the occupancy and use of that building can be even longer (many decades), a record of design intent (problem statements) along with the description of the design (solutions - the results of design decisions) can serve to aid the understanding of the many interested parties who come and go over the full course of a building's lifecycle: those who require and use design products; and, those who deliver them.

Where design intent is also modelled, validation of requirements can be done in a different manner to validation against third party codes and regulations. A rule in the regulatory model encodes the requirement logic within the rule itself, and external to the data model against which it is applied. In contrast, it is possible to define a rule for the continuous briefing/designing process as a "meta-rule" that works generically using the solution entities and the requirements that are each explicitly modelled in the source data. This is a subtle, but useful, distinction and an example will be given later, but first it is necessary to explain a way in which requirements might be modelled using IFC, and in order to do that it is necessary to understand the content of design briefs.

3. AN INVESTIGATION OF BRIEFING ARTEFACTS

Written briefs are a form of discourse between the brief writer and the designer/reader in the sense that design context and design intentions for the proposed project need to be conveyed. In order to dissect this material into its constituent generalized concepts, a relevant and applicable set of research tools to use are those under the broad umbrella of content analysis. Content analysis, in simple terms, is 'a general procedure for objectively identifying the characteristics of textual material' (Jones, 1996, p. 127).

Schreier (2012) describes content analysis as an iterative 'dialogue with cases'. For the research reported here, these source cases are forty-six real project briefs. However, the chosen documents are only a sample of the potential population of interest. Sampling is necessary for obvious reasons since it would be too onerous (and realistically impossible) to code all existing briefing material. The briefs were obtained by contacting briefing consultants, architects, and client representatives from Australia only, but in order to offset an overly Australian bias, briefs were also sourced from the websites of architectural professional organizations such as the Royal Institute of British Architects (RIBA) who publish competition briefs. Consequently, the selection of the briefs for analysis seeks to address: work taking place internationally; work related to different building types (health, education, workplace, and lifestyle); work at different scales (master planning, urban design, building design, interior design, and, in one case, product design); competitions and other types of project procurement; and, briefs written at different levels of detail (broad scope expressions of interest through to very detailed requirement specifications). All briefs were available in English, so it should be noted that there is some inherent bias in the sample relative to non-English speaking designers.

Content analysis is a reductionist methodology in the sense that it condenses the subject material by coding according to categories which may be derived inductively out of the text being analysed, or applied deductively from prior theoretical considerations outside of the text itself. In the analysis undertaken for this research both of these techniques are employed, the first automated via software, and the second done manually.

For the initial automated analysis the online version of the Leximancer text analytics software was used (Lexi-Portal, 2014). Each briefing file was assigned to one of four document folders according to its briefing type:

- Competition (seeking design submissions);
- Strategic ("visionary", broad objectives);
- Functional (more detailed, particularly concerning organizational and functional considerations);
- Comprehensive (very detailed, including room data sheet level briefing as well as functional and strategic requirements).

By dividing the briefs in this way the intention was to allow for cross-comparisons to be made between the subject matter of briefs created for various stages of the design process. Leximancer discovers the concepts occurring in source documents inductively. The steps a user of the software must follow are shown in figure 1.

Figure 1: Leximancer process

The software performs both conceptual analysis (identifying each distinct concept and the frequency at which it occurs), and relational analysis (identifying the main relationships between concepts). It produces interactive graphical maps to show the results of a "run", clustering related concepts as it has ranked them. Apart from the initial division of the briefs into four folders the analysis done using Leximancer is otherwise unmediated.

The use of Leximancer is an example of an inductive category development approach. However, the concepts derived in this way are not directly equivalent to the entities that are defined in the IFC international standard which is increasingly used for AEC projects. For the manual coding of the briefs, a deductive category development approach was first undertaken in which theoretical-based categories are adopted. In this case, the theoretical basis is the IFC schema. Basing the coding on those entities allows for any apparently missing constructs in IFC to be discovered inductively as part of the ensuing coding process.

Briefs on the whole are written in a formal manner, divided into clear sections that contain clauses which usually address one subject item per clause. For the manual coding, these subdivisions of the source material correspond well to what Schreier (2012, p. 131) describes as 'units of coding'. Whole clauses are predominantly used except where it is necessary to split a clause due to more than one concept being addressed. Each clause is further subdivided into "subject", "object", "relation", and "property" component parts to align with the first level of subclasses in IFC (as defined in all released versions) below the fundamental IfcRoot entity:

- IfcObjectDefinition for subjects and objects ("things" / "nouns");
- IfcPropertyDefinition (characteristics of entities);
- IfcRelationship (relationships between entities or between entities and associated property definitions).

Holsti (1969) describes five guidelines for constructing a set of categories:

1. Categories should reflect the purposes of the research;
2. Categories must be exhaustive (all of the material can be assigned to at least one category);
3. Categories should be mutually exclusive (that is each item in a text can only be coded against a single category, and, in particular, there should be no need for a catch-all "other" category);
4. Categories should be independent – the assignment of one item to a given category should not affect the assignment of other items;
5. Categories should be derived from a single classification principle – different levels of analysis are not mixed, for example, there is 'not a category for oranges plus a category for fruit'.

For coding of the subjects/objects and relationships, IFC entity categories are used plus further extended categories that are derived through the initial coding and category revision process. For coding of the properties, the concepts are allowed to emerge inductively through coding that identifies the subject matter of the property (usually the predominant word used in the text which best summarizes that property). This coding frame, in general, satisfies Holsti's five criteria except it should be noted that IFC, because of the nature of its sub-classing and inheritance mechanisms, from generic entities (for example, IfcOpening) down to more specific ones (for example, IfcWindow and IfcDoor) does violate the last criterion. The violation also emerges through a consideration of types relative to instances of entities in the progression from generic to specific as briefing and designing develop in more detail across stages of the design process. The contradiction is allowed to remain in order to preserve this insight in the reported analysis. The IfcProxy class which is used in IFC to deal with otherwise non-differentiated entities violates the third criterion since it is a miscellaneous catch-all mechanism; therefore it is not used as a coding category. The coding of each clause also uses additional codes to signify the strength of the desire expressed (increasing from wish to need, as indicated by the presence or absence of strength words such as "shall" or "must" in a clause), and to discriminate contextual statements (those describing the existing situation) from requirement statements (those describing some desired future state.

4. ANALYSES

4.1 Automated coding

The written project briefs were first converted to a common file format (PDF) and then grouped into four folders indicating the degree of resolution in the design process that each brief addressed. This file repository was then inputted to Leximancer for analysis. Figure 2 shows a concept map produced from 1000 iterations on the source data. The source folders are shown in black capitals, with concepts clustering at distances away from their sources, and each other, relative to their inter-relatedness. Concepts shown in hot colours (red being the "hottest") are those ranked with most importance. As colours move towards the other end of the spectrum, so their importance diminishes.

While not completely clear-cut, there is some discernible logic to be found in the grouping of the concepts on this map. Competition briefs contain considerable amounts of information concerned with the process and conditions of the competition. Organizations, people, and their activities tend to cluster in the strategic quadrant. Spaces and space types tend to cluster in the functional quadrant. Physical building elements tend to cluster in the comprehensive quadrant. This would seem to agree with a continuous model of the briefing process over time. Generically, briefs at different stages of a design process seem to progress from people-centric to space-centric to physical element-centric as Wade (1977) describes in his person-purpose-behaviour-function-object model.

Figure 2 shows how Leximancer can provide useful conceptual insights derived from a body of text in natural language terms. However, building information models are represented and exchanged using domain-specific "languages" such as IFC. This required an alternative approach to coding.

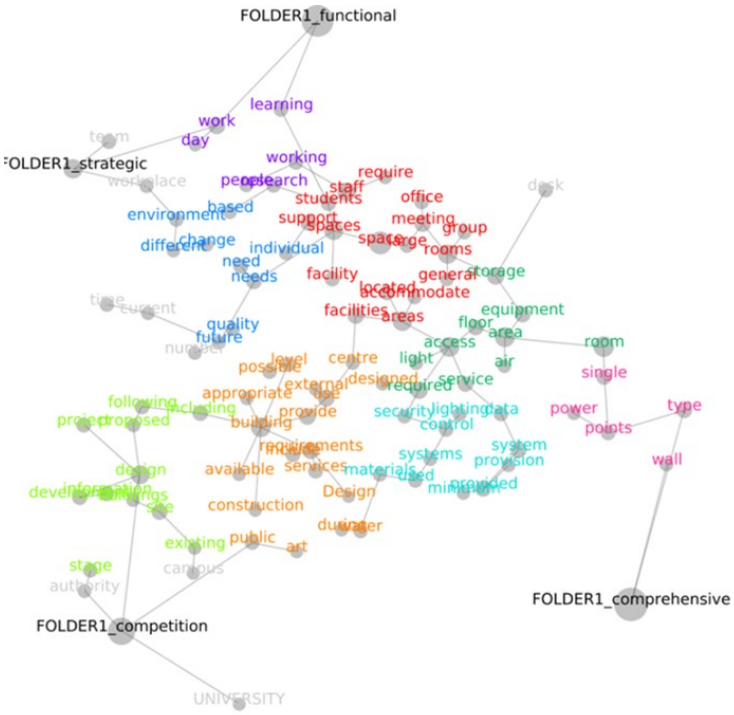

Figure 2: Unmediated analysis of sample briefs – concept map

4.2 Manual coding

Every brief in the sample set was coded manually clause-by-clause. At a meta-level, the clauses were each coded according to their intention ("requirement level"). A requirement level is categorized in terms of: level of need expressed ("vision" and "wish" versus a more tangible "need"); information about the current situation at the point in time at which the brief is written ("context"); a pointer to some other information external to the brief ("reference"); an example to inform the design ("exemplar"); a limiting factor ("constraint"); or a possible solution option ("proposal"). Figure 3 shows the results for this component of the overall coding.

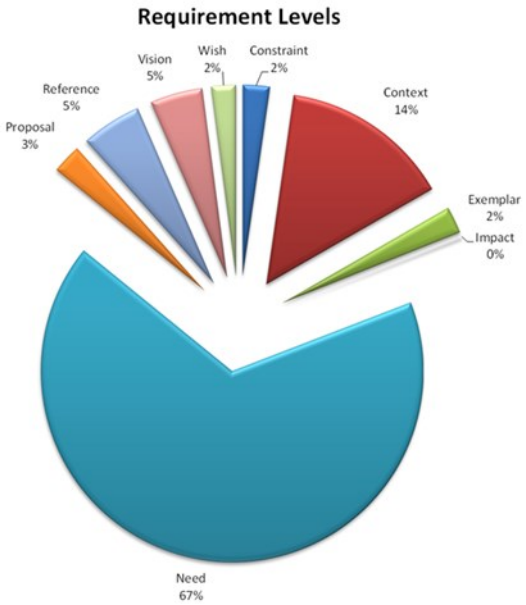

Figure 3: Manual coding – requirement level of a clause

While "need" is by far the most common interpretation of a clause in a brief, it can be seen from the graph that briefs also contain a significant quantity of contextual entities (24% if "exemplar", "reference", and "proposal" are counted alongside "context"). This finding is one indicator towards the idea that briefing (stating the problem) is a continuous process alongside, and intertwined with, designing (defining the solution). From an object-oriented perspective (BIM), decisions that have been made up to a given point in time find expression as object instances (an existing site, building, space, another object, or object property), and decisions that are to be made in the future are indicated either as descriptive statements (requirement properties attached to those existing contextual object instances), or by reference to generic type objects as placeholders for the final decision. The identity of these "instances" relative to currently defined IFC entities was coded in terms of the subjects and objects of a clause ("the nouns"), the properties associated with those subjects and objects, and the relationships between them.

The analyses for the manual coding were done according to the same grouping of briefs that was used for the Leximancer analysis, and the progression from people-centric to space-centric to physical element-centric was again observed. The chart shown in Figure 3 is an example of what was also produced for subjects, objects, properties, and relationships to show the results of the coding. The majority of items could be coded successfully as IFC entities, but it is from the identification of the exceptions that insights can be gained regarding potential extensions to IFC for briefing purposes.

The exceptions identified were of two distinct kinds: those related to a requirement relationship between entities; and, those related to the use of entity types to make generic statements of requirement.

As an example of a requirement expressed as a relationship, consider this statement from one of the briefs – 'The location of the kitchen to be adjacent to the café/bar area'. This is indicating a topological relationship between two proposed spaces. However, the statement is made in words and is pre-geometric because neither of these two spaces yet has form or location. The requirement could be expressed in IFC as an adjacency property in a property set associated with one of the two spaces but that would mean that there would be no corresponding inverse against the other space. Using a relationship entity that is linked to each of the two spaces to capture the adjacency requirement would seem to be a stronger means to capture the semantics of this statement. Furthermore, if that relationship is qualified by the degree of adjacency required, the "requirement level" of the clause can also be expressed. The adjacency requirement expressed as a relationship between entities may then later be checked geometrically once those entities are defined with geometric form.

Types were found to be used for briefing as generic placeholders for later instantiation of actual instances of the type, or as shorthand means to describe an existing situation. For example, here is a requirement for a quantity of

a particular space type – 'Provide 100 guestroom suites'. It is stated as an imperative; therefore the strength of the requirement can be understood as relatively strong. The target of the requirement (the other side of the relationship) is unstated. Most often in these situations, the implied target is the "project" as a catch-all entity. This example, therefore, shows a relationship for a required quantity between an instance entity (the project) and a type entity (the guestroom suite). If a statement uses the quantity of a type to describe an existing situation, again there is no corresponding IFC relationship entity that can be used. For example – 'The Division breakdown: General Manager (1), Senior Project Analysts (5), Project Analysts (6), Contractors (2), Support Staff (3)'. Here an organizational entity currently has a relationship with (contains numbers of) people in particular roles. The organization can be instantiated in IFC as an IfcActor, but for the person roles, there is no corresponding actor type in IFC. For this example, two possible extensions to IFC are indicated: the quantification relationship, and an IfcActorType entity.

If "requirement by quantity of type" and "requirement by adjacency" are expressed as requirement relationships, there is also a corresponding implication for two other relationship constructs that were found in the source material. Firstly, one way to state a requirement is by doing so using a qualifying property attached to an entity. For example, 'an executive office shall have an area of 18 square metres'. This can be done in IFC by including an area property in a property set linked to the space or space type entity using the IFC standard relationship for this purpose (IfcRelDefinesByProperties). The area property could be named as RequiredArea, or alternatively, the property set can include "Requirements" in its name. The BPie (2012) initiative has standardized some property sets for briefing using the property set naming methodology. However, an alternative construct could be to use a specialized requirement relationship to link property sets to object instances or types, In this case, instances of standardized property sets can be understood as representing actual property values ("defined-by-properties") or required property values ("required-by-properties") depending on the kind of relationship used to do the linking. Similarly, a requirement that is expressed as a reference to some source outside of the brief itself could be instantiated using a relationship. In this case, IFC already has an IfcRelAssociatesDocument relationship that could be used, but if the same idea of qualifying the relationship as a requirement is applied, then both the explicit notion of "requirement" and the "requirement level" attribute can be expressed. For example, in the following statement the requirement level is a strong "need" relative to the external reference document – 'the building shall be designed to comply with the Building Code of Australia'.

4.3 Parallelism between briefing and designing

In the situation where briefs and designs exist in separate repositories cross-checking and synchronization related to "briefed" versus "designed" entities relies on maintaining a common identifier for corresponding entity instances. For example, the brief contains a space instance with a user-defined code, and the design contains a corresponding space instance with the same user-defined code. This simple coding mechanism constitutes an implicit relationship which allows functionality such as checking the properties of the two space representations (as-briefed and as-designed) against each other, and where the entities contained by those spaces are also coded, checking the correspondences of the contained entities as well. A consideration of the same example in terms of an integrated brief/design repository reveals an apparent problem. If both instances co-exist in the same repository, then without some form of differentiation, an uninformed query to return all space instances and sum their areas will return the wrong total because it will include all space instances (regardless of whether they are as-required or as-designed) instead of filtering to retrieve one set or the other.

The discussion of properties in section 4.2 indicates one way in which the required properties of an entity instance can be maintained separately from its actual properties through the use of two distinct kinds of relationship. There is now the possibility for the requirement properties and designed properties to be contained in one repository without the need to duplicate the entity instance. Stating required and actual quantities of an entity using relationships is a further useful mechanism when we understand that the entity on the required side of the relationship must be a type. To illustrate this point, consider the distinction between an instance of a space and the instance of a space type in a data repository. Both are unique, but the space represents a concrete "thing" whereas the type is the definition for a "kind of thing". For example, the Executive Office type defines the characteristics of what the generic space may be, but it is only notional. It is not an instance such as the space named Room 101 that is a kind of Executive Office. In IFC, the relationship that defines "is kind of" between the two is an IfcRelDefinesByType. Using the proposed quantity relationships, various numbers of the Executive Office can be required by many organizational units without the need to instantiate any more than the one

instance of that type. When instances of actual spaces defined by this type are eventually created, there is no misunderstanding between what is briefed (the number as an attribute of the relationship and the type it refers to) and what is designed (the instances that are kinds of that type).

To give some indication of the possible reduction in redundancy using this type quantification methodology for briefing, consider the following partial example. The design brief for a hospital inpatient department requires a quantity of ten 4-bed wards. If this is expressed using an instance of IfcActor for the department, types for the ward (IfcSpaceType) and bed (IfcFurnitureType), collected together using the proposed required quantity relationship, the number of entities in the briefing model will be 5. That is, one for the IfcActor instance, plus 2 relationships, and 2 types. More generally, the formula for the number of entities to be instantiated can be expressed as:

$q1 + 2n$

Where q1 equals 1 and n is the number of nested levels below the root instance

On the other hand, if this example is expressed entirely using instances of the entities department (IfcActor), ward (IfcSpace), and bed (IfcFurniture) collected together using an IfcRelAggregates relationship, the number of entities needed to do the briefing will be 62. That is, one for the IfcActor instance, plus one IfcRelAggregates instance to collect the ten wards to the department, plus ten IfcSpace instances for the wards, plus ten IfcRelAggregates instances to collect each set of beds to their associated ward, plus forty IfcFurniture instances for the beds. In this case, the generalized formula for calculating the number of entities to be instantiated in a briefing model is:

$q1 + (q1 + (q1 \times q2)) + \ldots ((qn\text{-}2 \times qn\text{-}1) + (qn\text{-}1 \times qn))$

Where q1 equals 1,

n is the number of nested levels below the root instance,

and qn is the number of required instances at level n

In other words, at each level of an aggregated hierarchy of instances, the number of relationships will be equal to the number of instances in the level above, and the number of instances will be the required number of instances at this level multiplied by the number of instances at the parent level above.

For this simple example, there are roughly twelve times more entities instantiated using the instance methodology versus the type methodology. In a large project such as a hospital, where repetition of types recurs frequently, this proliferation of instantiated entities can be significant. On the other hand, where a project exhibits very little type repetition, the difference between the two methodologies will be negligible because the qn factors will mostly have a value of 1.

5. PROPOSED EXTENSIONS TO IFC FOR BRIEFING

The preceding two sections describe how relationships can be used to express briefing requirements that were found from the analysis of written briefing material. Also discussed is the use of types as placeholders for later creation of instances representing solutions to those generic requirements. Six new relations and one new type are therefore proposed as useful for representing briefing concepts:

Table 1: Proposed extensions to IFC for briefing

Relationships	
IfcRelRequires	An abstract subtype of IfcRelationship under which to collect the instantiable requirement relationships. This entity includes two optional attributes that have been adopted from the earlier Pset_ProductRequirements property set:

	• DemandImportanceValue to indicate the strength of a need; • SatisfactionValue to indicate the assessed satisfaction of the requirement.
IfcRelRequiresByProperties	A subtype of IfcRelRequires to aggregate an IfcProperytSet to an IfcObjectDefinition in which all the properties are specified as requirements.
IfcRelRequiresByDocument	A subtype of IfcRelRequires to handle the assignment of a document containing requirement information to object occurrences or object types.
IfcRelRequiresByAdjacency	A subtype of IfcRelRequires to handle the specification of a requirement for adjacency between objects or object types.
IfcRelRequiresByType	A subtype of IfcRelRequires to handle the aggregation of types by required quantity.
IfcQuantifiesByType	A subtype of IfcRelationship to handle the aggregation of types by quantity.
Types	
IfcActorType	The IfcActorType defines a list of commonly shared information for occurrences of actors.

The IFC4 (2013) release of the IFC specification has significantly increased (over previous releases) the number of types that are defined. IfcActorType is proposed here as a further addition that has applicability for briefing at a pre-spatial, people-centric phase. Another entity type that may be relevant to briefing (particularly if, and when, BIM is employed at an urban scale) is an IfcBuildingType. However, no explicit evidence for the use of this entity type was found in the analyses undertaken.Compliance checking

Having explained how briefing requirements can be modelled alongside solution entities (the design context at a point in time) using an extended version of IFC, it is now possible to show an example of how these extensions allow for the use of meta-rules for compliance checking of design models.

So, again consider the hospital inpatient example described in section 4.3. If the requirement for ten 4-bed wards is defined entirely in an external rule (say, for example, a ministry of health has decreed that all hospitals of a certain size must have this many wards as a minimum), then the rule can be written to use arguments that define the parameters of the requirement: the name and entity type of the target entity to be searched for (spatial zone type hospital inpatient unit); the entity type and name of the entities that are "owned" (space type 4-bed ward); and, the required number to be "owned" by this target entity (10);. The function within the rule needs to find and count ten (or more) instances of entities defined by the required type that have an "owned" relationship to the target entity.

In EXPRESS the rule can be encoded like this:

```
-- This function performs a requirement compliance test based on values provided by the
--   user for what to look for (the name and type of a "containing" object, plus the
--   quantity of a named type required).
-- The function checks 2 sets of possible objects:
-- The first set is objects that are the aggregation (decomposition) of the containing
--   object (aParent).
-- The second set is objects that are the nesting (decomposition) of the containing
object
--   (aParent).

QUERY_FUNCTION  CheckCount(parentName:STRING;  parentType:STRING;  childTypeName:STRING;
childType:STRING; nCheck:REAL) : BOOLEAN;

    LOCAL

        result       : BOOLEAN;

        aParent      : IfcObject;

        aInstance    : IfcObject;
```

```
        aTypeRel      : IfcRelDefinesByType;
        aAggRel       : IfcRelAggregates;
        aNestRel      : IfcRelNests;
        j, k, m       : INTEGER;
        nFound        : INTEGER;
    END_LOCAL;

    ON_ERROR_DO
        RETURN(?);
    END_ON_ERROR_DO;

    -- 1. Get the requested parent object (using parentType and parentName to locate it
    --    and initialise the check count.
    --    aParent is the container of the objects of interest, nFound is a counter for
    --    the objects that fulfil the requirement.
    nFound = 0;
    aParent := GetIfcObjectByCondition('Name = ''' + parentName + '''', parentType);

    -- 2. Loop through all objects contained by the parent object.
    -- 2.1. Check if an object (aInstance) of the required named type (childType and
    --      childTypeName) is a compliant object for the first set of objects
    --      (by aggregation).
    --      If the object inside the parent's decomposition is of the right named type
    --      this object fulfils the requirement, and therefore the compliant count is
    --      incremented (nFound).
    REPEAT j:=1 TO sizeof(aParent.IsDecomposedBy);
        aAggRel := aParent.IsDecomposedBy[j];
        REPEAT k:=1 TO sizeof(aAggRel.RelatedObjects);
            aInstance := aRel.RelatedObjects[k];

            REPEAT m:=1 TO sizeof(aInstance.IsTypedBy);
                aTypeRel := aInstance.IsTypedBy[m];

                -- Check that child is of the right named type (another function).
                IF CheckType(aTypeRel.RelatingType, childType, childTypeName) THEN
                    nFound = nFound + 1;
                END_IF;
            END_REPEAT;
        END_REPEAT;
    END_REPEAT;

    -- 2.2. Check if an object (aInstance) of the required named type (childType and
    --       childTypeName) is a compliant object for the second set of objects (by
nesting).
    -- If the object inside the parent's decomposition is of the right named type this
    --  object fulfils the requirement, and therefore the compliant count is incremented
    --  (nFound).
```

```
        REPEAT j:=1 TO sizeof(aParent.IsNestedBy);
            aNestRel := aParent.IsNestedBy[j];
            REPEAT k:=1 TO sizeof(aNestRel.RelatedObjects);
                aInstance := aRel.RelatedObjects[k];

                REPEAT m:=1 TO sizeof(aInstance.IsTypedBy);
                    aTypeRel := aInstance.IsTypedBy[m];

                    -- Check that child is of the right named type (another function).
                    IF CheckType(aTypeRel.RelatingType, childType, childTypeName) THEN
                        nFound = nFound + 1;
                    END_IF;
                END_REPEAT;
            END_REPEAT;
        END_REPEAT;

        -- 3. Compare the number of compliant objects found (nFound) against the requirement
        --    (nCheck).
        --    Returns true if the number of compliant objects found is greater than or equal
        --    to the requirement, false otherwise.
        IF nFound >= nCheck THEN
            result := TRUE;
        ELSE
            result := FALSE;
        END_IF;

        RETURN(result);
    END_QUERY_FUNCTION;
```

In the above rule, because the arguments are all handed in as variables to be operated on, the requirement is therefore entirely defined inside the functional code.

Now consider the same example using the IfcRelRequiresByType extension. In this case, the requirement is explicitly stated in the data model, so it is only necessary to hand in the unique identifier of that requirement.

In EXPRESS the rule can now be encoded like this:

```
    -- This function performs a generic requirement compliance test against the proposed
    --   relationship information using types (from IfcRelRequiresByType).
    -- The function checks 2 sets of possible objects:
    -- The first set is objects that are the aggregation (decomposition) of the relating
    --   object by the Type (aParent).
    -- The second set is objects that are the nesting (decomposition) of the relating object
    --   by the Type (aParent).
    QUERY_FUNCTION CheckCountUsingRequirementByType(relReqt:IfcRelRequiresByType) : BOOLEAN;
        LOCAL
```

```
        result       : BOOLEAN;
        aParent      : IfcObjectDefinition;
        aType        : IfcTypeObject;
        aInstance    : IfcObject;
        aTypeRel     : IfcRelDefinesByType;
        aAggRel      : IfcRelAggregates;
        aNestRel     : IfcRelNests;
        j, k, m, p   : INTEGER;
        nFound       : INTEGER;
        nCheck       : REAL;
    END_LOCAL;

    ON_ERROR_DO
        RETURN(?);
    END_ON_ERROR_DO;

    -- 1. Initialize values using the proposed IfcRelRequiresByType relationship object.
    --    aParent (in this case the Inpatient Unit) is the container of the objects of
    --     interest (instances of aType - in this case a 4-bed ward space type), nFound
is
    --      a counter for the objects that fulfil the requirement, and nCheck is the
number
    --    of objects that must be met as part of the requirement.
    nFound = 0;
    aParent := relReqt.RelatingObject;
    aType := relReqt.RelatedType;
    nCheck := relReqt.RelatedQuantity;

    -- 2. Loop through each instance of the required type.
    -- 2.1. Loop through each "defines by" relationship of the required type.
    REPEAT j:=1 TO sizeof(aType.Types);
        aTypeRel := aType.Types[j];

        -- 2.1.1. Loop through each instance (aInstance) on the "defined by" side of
        --        the relationship.
        REPEAT k:=1 TO sizeof(aTypeRel.RelatedObjects);
            aInstance := aTypeRel.RelatedObjects[k];

            -- 2.1.1.1. Check if this object (aInstance) of the required type (aType)
            --    is a compliant object for the first set of objects (by aggregation).
            --    If the object inside the parent's decomposition matches the instance
            --    of the required type (aInstance), this object (space) fulfils the
            --    requirement, and therefore the compliant space count is incremented
            --    (nFound).
            REPEAT m:=1 TO sizeof(aParent.IsDecomposedBy);
                aAggRel := aParent.IsDecomposedBy[m];
                REPEAT p:=1 TO sizeof(aAggRel.RelatedObjects);
```

```
                        IF aAggRel.RelatedObjects[p] = aInstance THEN
                            nFound = nFound + 1;
                        END_IF;
                    END_REPEAT;
                END_REPEAT;

                -- 2.1.1.2. Check if this object (aInstance) of the required type (aType)
                --    is a compliant object for the second set of objects (by nesting).
                --    If the object inside the parent's nesting decomposition matches the
                --      instance of the required type (aInstance), this object (space)
fulfils
                --       the requirement, and therefore the compliant space count is
incremented
                --   (nFound).
                REPEAT m:=1 TO sizeof(aParent.IsNestedBy);
                    aNestRel := aParent.IsNestedBy[m];
                    REPEAT p:=1 TO sizeof(aNestRel.RelatedObjects);
                        IF aNestRel.RelatedObjects[p] = aInstance THEN
                            nFound = nFound + 1;
                        END_IF;
                    END_REPEAT;
                END_REPEAT;
            END_REPEAT;
        END_REPEAT;

        -- 3. Compare the number of compliant objects found (nFound) against the requirement
        --    (nCheck).
        --    Returns true if the number of compliant objects found is greater than or equal
        --    to the requirement, false otherwise.
        IF nFound >= nCheck THEN
            result := TRUE;
        ELSE
            result := FALSE;
        END_IF;

        RETURN(result);
    END_QUERY_FUNCTION;
```

As shown, the first code sample relies on its argument list to define the requirement. There is no explicit definition of the requirement in the data model itself. The second code sample, on the other hand, provides the functionality for determining compliance because the requirement is now explicit in the data model - "requiresByType" (quantity 10) is a requirement relationship linking inpatient unit to the 4-bed ward.

Furthermore, this function can be called from another function that performs a loop through all the requirements by type in the data model, thus checking everything relevant in one pass. Therefore, compliance functions can be separately packaged and applied to any data model that uses the generic requirement extensions for IFC because those requirements have been expressed using a standardised syntax and are in the data being checked.

6. CONCLUSION

The proposals made here add complementary functionality relative to related work that has gone before, both in data modelling for briefing and in compliance checking. The use of IfcRelRequires and its relationship subclasses clarifies the distinction between briefed and designed, while also allowing for a connection between the two. Additionally, the use of types in the progression from generic to specific is revealed to be an important component of the briefing/designing process. Statements of intent can now be modelled using the same data modelling schema as the design solutions, and these two aspects of the briefing/designing process can exist unambiguously alongside each other providing a foundation for a different, and more semantically rich, input to validation (meta-) rules.

The requirement relationships provide the ability to record many, and various, briefing requirements. This paper has briefly described how the requirements "by-adjacency" and "by-property" can be checked for compliance. A fuller example has been shown for the "quantity-of-a-type" requirement relationship. However, these methods for representing requirements also allow for recording requirements (such as vision statements) where there is no automated method to check compliance other than through human interpretation. At least, in this latter case, the requirements are there in the data, linked to the "requiring" entities, and alongside the solutions (the entities that represent the design).

In order to implement an integrated data model such as the one described, it must be possible to include the requirement relationships. These are links between "requiring" and "required" entities in the model, using the globally unique identifiers of those entities on each side of the relationship. The emergence and use of building model servers that house all of the various stakeholders' data in one consolidated and shared repository is the most direct way in which linking such as this can be accomplished. However, where the data is housed as a federation of discrete data models (owned by different stakeholders), then some further means will be required to allow relationships to link across from an entity in one model to an entity (or entities) in another model. This is functionality beyond the scope of this paper.

7. REFERENCES

AR-5 (2006). IFC Extension Project: AR-5 Information Requirements Specification – [AR-5] Early Design – V5 (project leader Francois Grobler), A report by IAI NA and UK

Balachandran M., Rosenman M. A. and Gero J. S. (1991). A knowledge-based approach to the automatic verification of designs from CAD databases, in Gero J. S. (ed.), Artificial Intelligence in Design '91, Butterworth-Heinemann, Oxford, pp. 757-781.

BIM Industry Working Group (2011). BiM Management for value, cost and carbon improvement: A report for the Government Construction Client Group

Blyth, A. and Worthington, J. (2001). Managing the Brief for Better Design, Spon Press, London and New York

BPie (2012). Building Programming information exchange,

- General information http://www.nibs.org/?page=bsa_bpie
- Requirements analysis http://projects.buildingsmartalliance.org/files/?artifact_id=4700
- Exchange requirements http://projects.buildingsmartalliance.org/files/?artifact_id=4699

buildingSMART 2014, Technical Vision, http://www.buildingsmart.org/standards/technical-vision

Ding, L., Drogemuller, R., Rosenman, M., Marchant, D. and Gero, J. (2006). Automating code checking for building designs-DesignCheck, CRC for Construction Innovation for Icon. Net Pty Ltd, Sydney, Australia

Fenves S. J., Garrett J. H., Kiliccote H., Law K. H. and Reed K. A. (1995). Computer representations of design standards and building codes: U.S. perspective, The International Journal of Construction Information Technology, Vol 3, No 1, pp. 13-34.

Gero J. S. (1982). A self-checking database for the Australian Building Code, CAD82, Butterworths, Guildford, pp. 119-125.

Holsti, O. R. (1969). Content analysis for the social sciences and humanities, Addison-Wesley,

Reading Massachussets

IFC2x3 (2006). Industry Foundation Classes IFC2x Edition 3,

Available electronically at http://www.buildingsmart-tech.org/specifications/ifc-releases/ifc2x3-release

IFC4 (2013). Industry Foundation Classes IFC4 Official Release,

Available electronically at http://www.buildingsmart-tech.org/ifc/IFC4/final/html/index.htm

ISO 10303-11 (2004), Industrial automation systems and integration -- Product data representation and exchange -- Part 11: Description methods: The EXPRESS language reference manual

ISO 16739 (2013), Industry Foundation Classes (IFC) for data sharing in the construction and facility management industries. ISO TC184 SC4.

Jones, R. (1996). Research Methods in the Social and Behavioral Sciences, Second Edition,

Sinauer Associates Inc, Massachussets

Kiviniemi, A. (2005). Requirements Management Interface to Building Product Models, PhD Thesis,

Stanford University, California

Krippendorff, K. and Bock, M. A. (2009). The Content Analysis Reader, SAGE Publications

Lexi-Portal (2014). Leximancer software (online Version 4).

Maissa S., Frachet J. P., Lombardo J. C., Bourdeau M. and Soubra S. (2002). Regulation checking in a virtual building, CIB w78 conference 2002.

Marchant, D. (2015). Capturing and integrating the design brief in building information models, PhD thesis, UNSW Australia

Niemeijer, R.A., Vries, B.D. and Beetz, J. (2009). Check-mate: automatic constraint checking of IFC models. Managing IT in construction/managing construction for tomorrow, pp.479-486.

PAMPeR (2004). IFC Extension Project: FM-9 Portfolio and Asset Management - Performance Requirements (project leader Gerald Davis),

Pauwels, P. and Zhang, S., (2015). Semantic rule-checking for regulation compliance checking: an overview of strategies and approaches. In 32rd international CIB W78 conference.

Pena, W. with Caudill, W. and Focke, J. (1977). Problem Seeking: An Architectural Programming Primer, Cahners Books International, Boston

Rosenman M. A., Gero J. S. and Oxman R. (1986). An expert system for design codes and design rules, in Sriram D. and Adey R. (eds), Applications of Artificial Intelligence to Engineering Problems, Springer-Verlag, Berlin, pp. 745-758.

Schreier, M. (2012). Qualitative content analysis in practice. Sage Publications

Solihin W. (2004). Achieving automated code checking with ease, a white paper, NovaCITYNETS Pte Ltd, Singapore.

Wade, J. (1977). Architecture, Problems and Purposes, Wiley-Interscience, New York

Woodbury R., Burrow A., Drogemuller R. and Datta S. (2000). Code checking by representation comparison, CAADRIA: Proceedings of the 5th Conference on Computer Aided Architectural Design Research in Asia, pp. 235-244, CASA, Singapore.