博士学位論文

BIM/CIM における情報共有を目的としたプロダクトモデルとデータベースの適合性に関する研究

四月朔日 勉

2017 年 7 月

大阪大学大学院工学研究科

目次

第 1 章

序論

1.1　背景

1.1.1　BIM/CIM の状況

従来は，建築物の設計，施工は 2 次元の設計図を用いて行なわれていたが，コンピュータグラフィックス技術の進歩と，コンピュータの処理能力の向上に伴ない，対象の建築物を 3 次元プロダクトモデルで表現し，計画，設計，施工，維持管理のライフサイクル全体で利活用し生産性を向上させる試みが，欧米や先進国で行なわれるようになってきた。この試みは，BIM (Building Information Modeling) と呼ばれている。建築物を対象とした BIM に少し遅れて，道路や橋梁等の土木構造物に関しても同様の試みがなされており，この試みは，BIM for Infrastracture，米国では，CIM (Civil Integrated Management) と呼ばれている [1]．この概念図を図 1.1 に示す．

日本国内においても，欧米や先進国に影響を受ける形で BIM への取り組みが始まった．2009

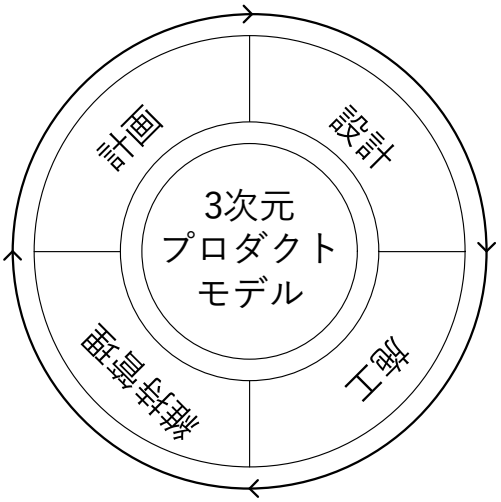

図 1.1　BIM/CIM の概念図

年の国土交通省による BIM 導入宣言が契機となり，国土交通省発注案件に BIM 発注が取り入れられるようになってきている。また，道路，橋梁等土木構造物の設計，施工，維持管理等に BIM の手法を取り入れて生産性を上げる試みを，国土交通省が平成 24 年度から CIM[*1](Construction Information Modeling) と称して開始している [2].

英国では，BIM を段階的に導入できるように，BIM レベルチャート（図 1.2）が公表されている [3]．このチャートは，BIM の成熟度が上がるにつれ，情報共有の手段が，紙 (Level 0) から，ファイル (Level 1)，ファイルとローカルデータベース (Level 2)，ウェブ上で統合化されたシステム (Level 3) へと変化していくことを表している．英国では，2016 年までに Level 2，2025 年までに Level3 を達成するという目標で BIM/CIM を実施している [4]．このモデルは英国の BIM に特化しているわけではなく一般的なモデルであり，日本における BIM/CIM もこのように進化していくと考えられる．現状では，日本は Level 0–1，最も BIM/CIM が進んでいる英国やフィンランドでも Level 2 であり，Level 3 に達している国はない．

1.1.2 BIM/CIM における情報共有

BIM レベルチャートからも分かるように，全世界的に，BIM/CIM の当面の目標は，インターネットを介した関係者間における情報共有，および，共同作業を実現すること，であると言える．しかしながら，BIM/CIM の対象である建築，建設分野には以下のような特徴があるため，他分野における情報共有よりも実現が困難であると考える．

1. 共有対象となるデータ量が膨大であること

   BIM/CIM では，対象とする建築物，土木構造物のあらゆる情報を，3 次元プロダクトモデル[*2]に集約して共有することを目指している．これには，建築物を構成する壁やドアの形状はもちろんのこと，材料や型番等の属性も含まれるため，建築物，土木構造物のあらゆる部材をモデル化することになり，プロダクトモデルのデータ量は非常に大きくなる．また，対象となる建築物，土木構造物の寿命が長く，短いものでも 30 年前後，長いものだと 100 年を越す場合もあり，ライフサイクルが進むほど多くのデータが発生し，それらがプロダクトモデルに蓄積されるため，さらに肥大化することになる．
2. 様々な関係者間で共有する必要があること

   大規模な建築物，土木構造物を対象とする場合は，そのライフサイクルには，施主，発注者，設計者，施工者を始めとした，非常に多くの関係者が関わる．日本では土木構造物のライフサイクルの各フェーズに関わる作業は，国土交通省が民間企業に発注して実施される場

[*1] 米国における CIM と略語は同じであるが，元となる用語が異なる．しかし，両者のコンセプトや目的，主に用いる手法はほぼ同様と捉えてかまわないと思われる．いずれにせよ，現在は，BIM，CIM は共にまだ過渡期であり，厳密な定義が定まっているわけではない．単に CIM と表記した場合は，海外，日本国内の区別なく，“土木構造物を対象とした BIM” の意味を表すものとする．

[*2] プロダクトモデルは対象物の特徴を抽象化した概念を定義したもの（スキーマ）の意味で用いられる場合と，対象物そのものをモデル化したもの（オブジェクト，またはインスタンスの集合）の意味で用いられる場合があるが，本論では後者の意味で用いる．

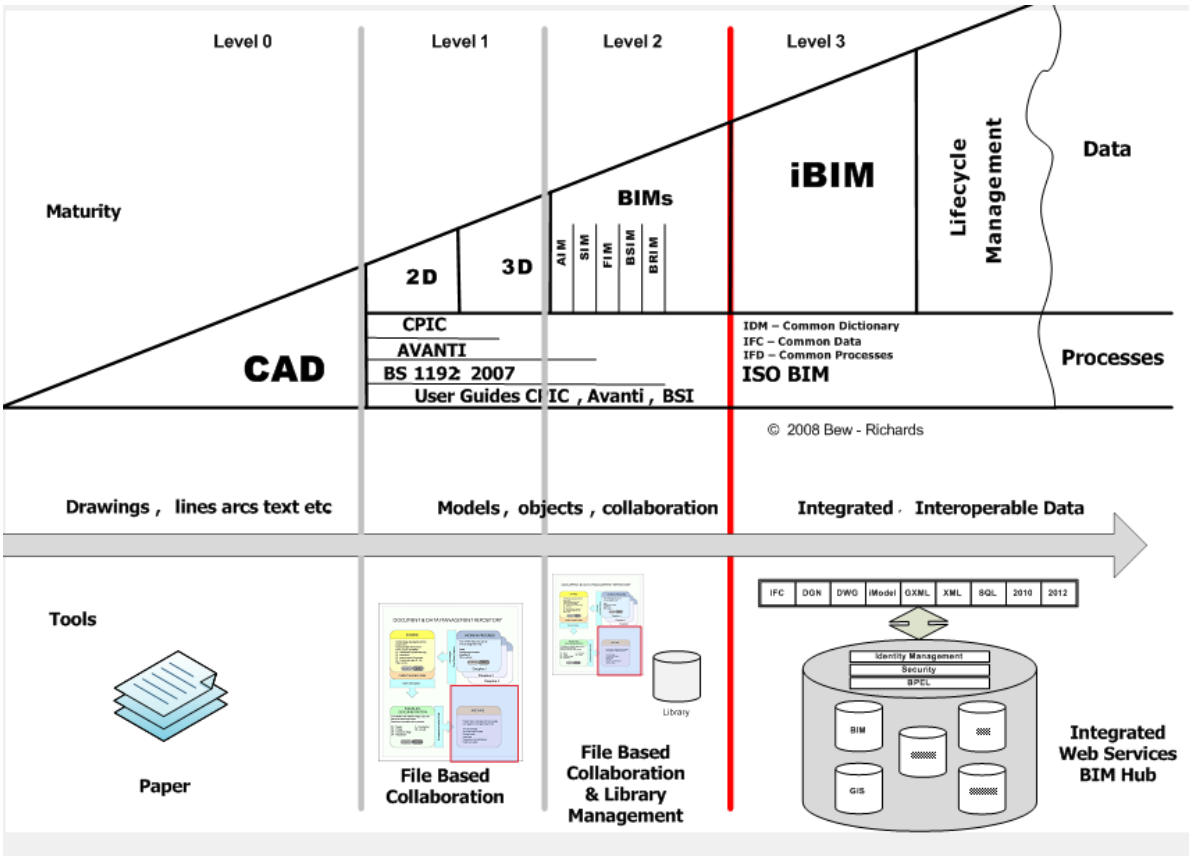

図 1.2　BIM レベルチャートと情報共有手法の関係（出典:A report for the Government Construction Client Group）

合がほとんどであるため，単に多数の関係者が関わるだけでなく，様々な組織に属した関係者が関わることにも留意する必要がある．

3. 様々な種類のデータを共有する必要があること

   BIM の対象となる建築物のほとんどが，ビルや住宅等であるのと異なり，CIM では，橋梁，トンネル，道路，ダム等様々な土木構造物が共有対象となる．当然，土木構造物の種類が異なれば部材の種類や取り得る属性の種類も異なるため，様々な種類のデータに対応せざるを得ず，システム化の難易度は高くなる．

このような背景から，BIM/CIM における共同作業を実現するには，組織の枠を超えた多数の関係者間で，膨大なデータを含む様々な種類のプロダクトモデルを共有する必要があることが分かる．これが，BIM/CIM における情報共有のあるべき姿であると考える（図 1.3）．

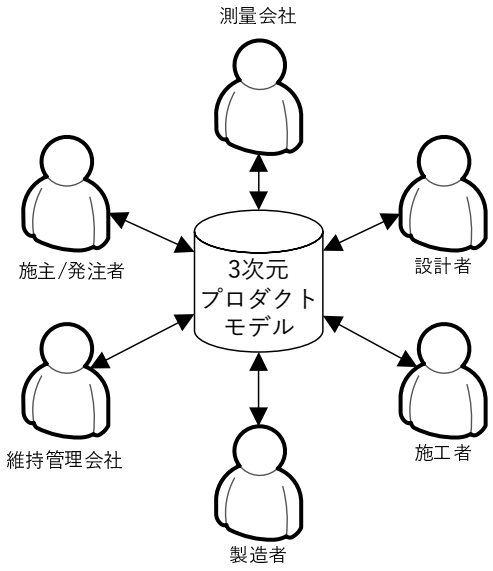

図 1.3 BIM/CIM における情報共有のあるべき姿

1.2 目的

前節で示した，“BIM/CIM における情報共有のあるべき姿” を実現するためのシステム（以下，BIM/CIM 情報共有システムとする）の構築には，3 次元グラフィックス，ネットワーク，データベース等に関連した様々な技術が必要になるが，それらは各分野で十分に研究，実用化されており，既存技術を組み合わせることにより BIM/CIM 情報共有システムを実現することは，十分可能であると考えられる．

しかしながら，共有対象となる建築物，土木構造物のプロダクトモデルには，ライフサイクル全般で発生する膨大な量の情報が含まれることになるため，円滑な情報共有の実現には，巨大なプロダクトモデルを効率良く取り扱う技術が必要になる．従って，BIM/CIM 情報共有システムを実現するためには，データマネジメント技術が最も重要であると考えた．

よって，本研究では，BIM/CIM 情報共有システムを構築すること想定して，BIM/CIM で標準的に用いられている IFC (Industry Foundation Classes) [5] と呼ばれるスキーマ[*3]で記述されたプロダクトモデル（以下，IFC プロダクトモデルとする）の格納に適切なデータベースとデータ構造を明らかにすること，つまり，IFC プロダクトモデルとデータベースの適合性を明らかにすることを主な目的とし，さらに，データベースに格納された IFC プロダクトモデルに対する，部分抽出等のデータマネジメント手法を開発することも目的とする．

[*3] モデルが取り得るデータ構造や，属性のデータ型や制約を定義したもの．

1.3 既往の研究

本研究に関連する既往の研究，または既往のシステムを挙げ，本研究の位置付けについて述べる．

1.3.1 IFC プロダクトモデルの共有に関する研究

前述したように，プロダクトモデルをネットワーク上で共有することは，BIM/CIM における作業を著しく効率化すると考えられているため，IFC プロダクトモデルの共有に関する研究が，様々な研究者によって行なわれている．

Faraj らは, WISPER というシステムを開発するにあたり, IFC プロダクトモデルを ObjectStore [6] という商用オブジェクト指向 DBMS[*4] (DataBase Management System) に格納している．そのため，IFC プロダクトモデルをオブジェクト指向プログラミング言語である Java 言語によりオブジェクト化したものを，データ変換せずに，そのまま DBMS に格納することが可能となっている [7]．また，足達は IFC Model Server の開発において，IFC スキーマをデータベーススキーマに自動変換し，システム開発作業を効率化する手法を確立している [8]．これにより，リレーショナル DBMS と XML DBMS の両方式が利用可能なシステムを実現している．これらは，IFC の初期（2000 年頃）の試みであるため，システム化自体を主目的としており，システムの性能やデータマネジメント技術に関してはそれほど着目していない．

近年では，Das らが Social BIMCloud [9] を，Beetz らが BIM Server [10] を開発している．Social BIMCloud は Apache Cassandra [11] というカラム指向型 DBMS が使われており，また，BIM Server は Oracle Berkley DB [12] というキー・バリューストア DBMS が採用されている．どちらも，NoSQL に分類される比較的単純なデータモデルを採用する DBMS であり，複雑なデータ構造を有する IFC プロダクトモデルを格納するため，内部のデータ構造に工夫を施している．Das らは，Social BIMCloud で NoSQL DBMS を採用した理由として，リレーショナルデータベースと異なり，格納データの構造を動的に変更可能である，スキーマレスなデータベースであることを挙げており，IFC プロダクトモデルの共有において，リレーショナルデータベースよりも NoSQL の方にメリットが大きいことを示唆しているが，複数のデータベース方式の比較検討を実施しているわけではない．

1.3.2 IFC プロダクトモデルのデータマネジメント技術に関する研究

Won らは IFC プロダクトモデルから必要な部分のみを抽出するアルゴリズムを考案している [13]．また，Arthaud らは 2 つのプロダクトモデルを比較し，各要素の変更，削除，追加，移動，移動かつ変更を検出することに成功している [14]．しかしながら，これら研究は，ファイル

[*4] 本論では，実際に動作するソフトウェアを指す場合は DBMS (DataBase Management System)，データベース方式や概念を指す場合はデータベースと記述する．

ベースの IFC プロダクトモデルを対象としており，DBMS に格納されたモデルには適用しづらい．BIM/CIM 情報共有システムの実現には，IFC プロダクトモデルを DBMS に格納する必要があるため，DBMS に格納された状態で適用可能な，部分抽出や差分抽出といったデータマネジメント技術を考案する必要がある．

1.3.3 NoSQL データベースに関する研究

NoSQL データベースに関しては様々な研究が行なわれているが，本論におけるデータベースの利用目的はプロダクトモデルの格納であるため，NoSQL データベースが取り扱うデータに着目して既往の研究や事例を整理する．

川谷らは，ドキュメント指向 DBMS である MongoDB [15] を利用して，センサデータの蓄積を実現している [16]．また，桑野は，株式会社サイバーエージェントが運営している Web サービスであるビグライフ[*5]で MongoDB を用いており，扱うデータの単位が大き過ぎないこと，読み込みに対して書き込みが少ないこと，といった MongoDB が得意とするユースケースを明らかにしている [17]．

また，ソーシャルネットワークサービスである LinkedIn[*6]では，ユーザ同士の関係を保持，分析するためにグラフ DBMS である Neo4j [18] が利用されている [19, 20]．

これらから，NoSQL データベースは，比較的小さな構造のデータ（数個〜多くとも数 100 個程度のインスタンスからなるデータ）を大量（数 100 万〜数億個）に処理するケースで利用されていることが分かる．実施されている研究もそのようなケースを想定したものが多く，本研究が対象とするプロダクトモデルのように，大きな構造のデータ（数 10 万以上のインスタンスからなるデータ）を対象とした研究は見当たらない．

1.3.4 本研究の位置付け

前述したように，IFC プロダクトモデルの共有に関する研究では，リレーショナルデータベース以外の様々な DBMS を使ったシステムが提案されている．BIM/CIM に関係のない一般的なシステム開発において，リレーショナル DBMS が採用される割合が大きいことを考えると，多くの研究者が，IFC プロダクトモデルにリレーショナル DBMS は向いていない，と考えていると想像できるが，NoSQL を含めた様々なデータベース方式を比較検討した研究は見当たらない．

現時点において，NoSQL データベースが利用されるケースは，前項で示したような，「小規模なデータを大量に処理する」ケースがほとんどであり，本論が目的としている， 建築物や土木構造物等の大きな構造のデータを少量（数千〜数万程度）処理するケースにおける研究は不十分であると言える．

また，データマネジメント技術に関する研究では，ファイルベース，またはオンメモリ状態にお

[*5] https://life.pigg.ameba.jp/
[*6] https://www.linkedin.com/

ける手法が論じられているに留まっており，DBMS に格納した状態でのデータマネジメント手法は十分に研究されていない状況である．

よって，“IFC プロダクトモデルとデータベースの適合性を明らかにする”，という本研究の主目的が達せられたとすれば，BIM/CIM 情報共有システムにおけるデータベースレイヤーの構築手法に一定の指針ができることになる．さらに，IFC プロダクトモデルを DBMS に格納した状態でのデータマネジメント技術を研究し，部分抽出等のアルゴリズムを開発することにより，BIM/CIM 情報共有システムの機能面における利便性や処理性能を向上させることが可能になる．

1.4 本論文の構成

本論文は，図 1.4 に示す通り，全 7 章で構成されている．

第 1 章では，本論文の研究背景と研究目的，既往の研究に対する位置付けについて述べた．

第 2 章では，BIM/CIM で標準的に用いられている IFC スキーマと IFC スキーマによって記述された IFC プロダクトモデルについて述べ，IFC プロダクトモデルのデータ構造の特徴を明らかにする．

第 3 章では，コンピューティングシステム開発におけるリレーショナルデータベース全盛の状況から，他方式のデータベースである，XML データベースや NoSQL データベースが開発された歴史的経緯を示す．その後，各種データベースとそのデータモデルについて説明し，リレーショナルデータベースと他方式データベースとを対比することで，それぞれのデータベースの特徴を明らかにする．

第 4 章では，IFC プロダクトモデルを，第 3 章で説明した主要なデータモデルに変換する手法を考案する．さらに，実験を通して IFC プロダクトモデルを格納するのに適したデータベース方式を明らかにする．

第 5 章では，IFC プロダクトモデルの管理に，NoSQL の一種であるグラフデータベースを利用した場合のデータマネジメント手法，特に部分抽出手法について論じる．さらに，グラフ DBMS に IFC プロダクトモデルを格納し，実際に部分抽出処理が可能であることを検証する．

第 6 章では，グラフデータベースを用いた BIM/CIM 情報共有システムを提案し，システムの開発コスト，メンテナンス性について考察する．

第 7 章で，本研究の結論と今後の課題について述べる．

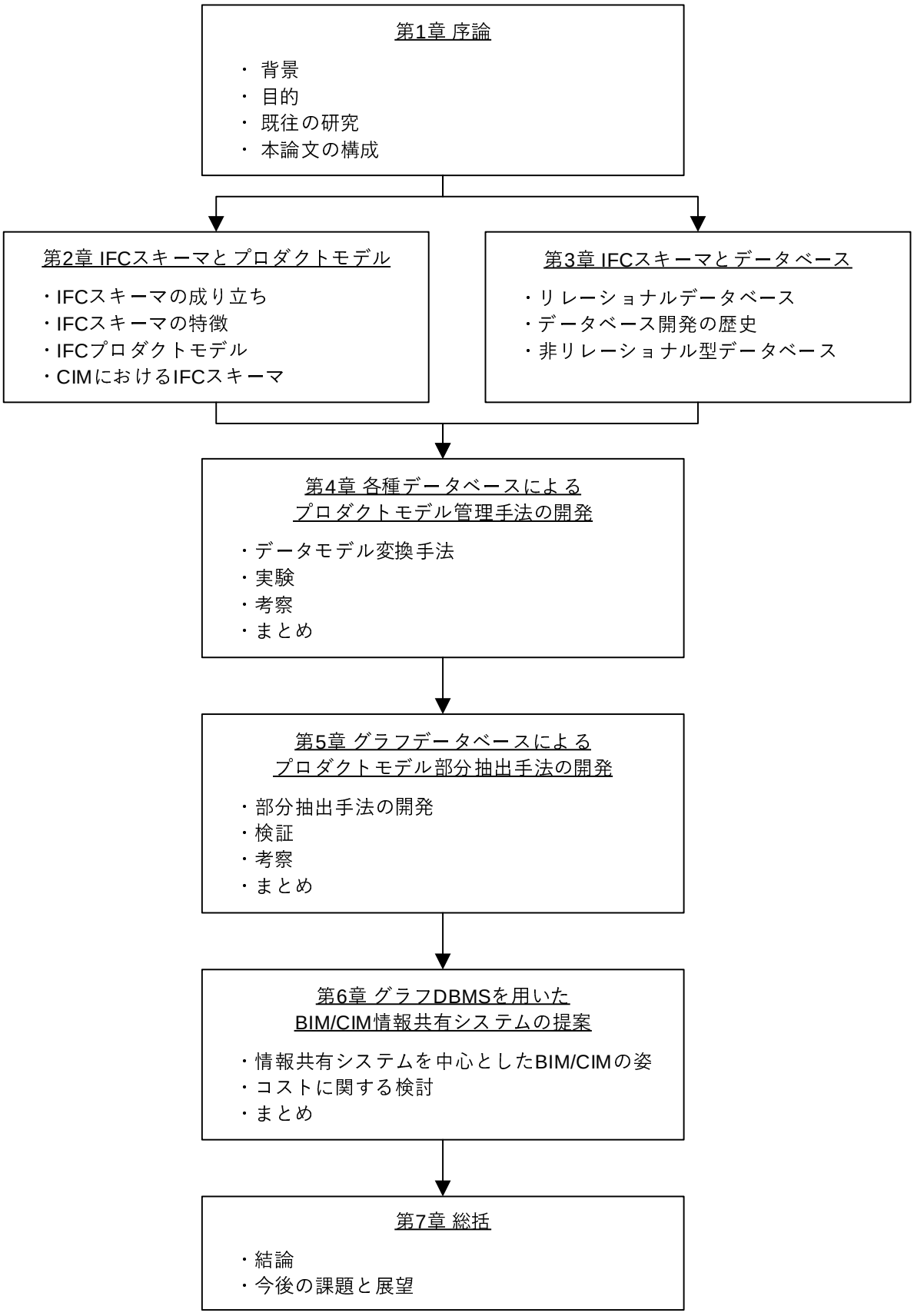
第1章 序論
・背景
・目的
・既往の研究
・本論文の構成
第2章 IFCスキーマとプロダクトモデル
・IFCスキーマの成り立ち
・IFCスキーマの特徴
・IFCプロダクトモデル
・CIMにおけるIFCスキーマ
第3章 IFCスキーマとデータベース
・リレーショナルデータベース
・データベース開発の歴史
・非リレーショナル型データベース
第4章 各種データベースによる
プロダクトモデル管理手法の開発
・データモデル変換手法
・実験
・考察
・まとめ
第5章 グラフデータベースによる
プロダクトモデル部分抽出手法の開発
・部分抽出手法の開発
・検証
・考察
・まとめ
第6章 グラフDBMSを用いた
BIM/CIM情報共有システムの提案
・情報共有システムを中心としたBIM/CIMの姿
・コストに関する検討
・まとめ
第7章 総括
・結論
・今後の課題と展望

図 1.4　本論文の構成

1.5 表記法

本論文では，IFC スキーマや IFC プロダクトモデルの図を UML (Unified Modeling Language) [21] で表記することにする．UML には様々な図 (Diagram) が用意されているが，本論文で使用するのは，クラス図 (Class Diagram) とオブジェクト図 (Object Diagram) である．それぞれの表記法を，本論文内で使用するものに限って，図 1.5，図 1.6 に示す [22].

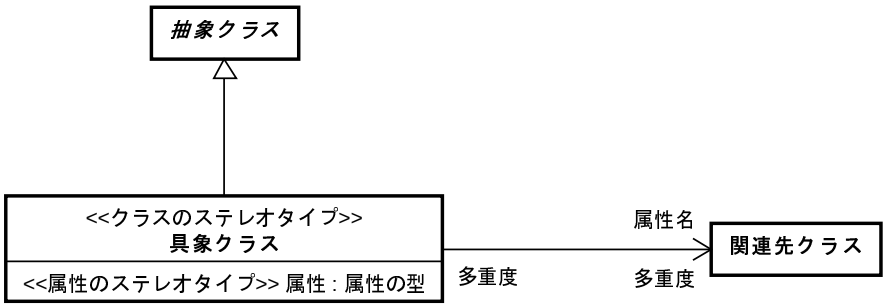

図 1.5　UML クラス図の表記法

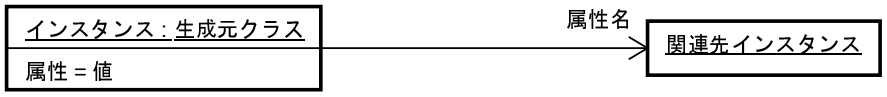

図 1.6　UML オブジェクト図の表記法

参考文献

[1] 矢吹 信喜: CIM 入門, 理工図書, 2016

[2] 国土交通省: CIM の概要, http://www.mlit.go.jp/tec/it/pdf/cimnogaiyou.pdf (参照 2017/6/1)

[3] BIM Workfing Party: A report for the Government Construction Client Group, 2011, http://www.bimtaskgroup.org/wp-content/uploads/2012/03/BIS-BIM-strategy-Report.pdf (参照 2007/6/1).

[4] 矢吹 信喜: CIM 入門, 理工図書, pp.181–183, 2016.

[5] buildingSMART International: Industry Foundation Classes IFC4 Official Release, 2013, http://www.buildingsmart-tech.org/ifc/IFC4/final/html/ (参照 2017/6/1).

[6] Ignite Technologies: ObjectStore®standard edition the database behind the world's most scalable applications, https://www.ignitetech.com/solutions/information-technology/objectstore/ (参照 2017/6/1).

[7] I. Faraj, M. Alshawi, G. Aouad, T. Child, J. Underwood: An industry foundation classes Web-based collaborative construction computer environment: WISPER, Automation in Construction 10, pp.79–99, 2000.

[8] Yoshinobu Adachi, Outline of IFC Model Server Development Project, VTTTEC-ADA, 2001.

[9] Moumita Das, Jack CP Cheng, Srinath S Kumar: Social BIMCloud: a distributed cloud-based BIM platform for object-based lifecycle information exchange, 2015.

[10] Jakob Beetz, Léon van Berlo, Ruben de Laat, Pim van den Helm: Bimserver.org – an Open Source IFC model server, Proceedings of the CIB W78 2010: 27 th International Conference, 2010.

[11] Apache Software Foundation: What is Cassandra?, http://cassandra.apache.org (参照 2017/6/1).

[12] Oracle: Oracle Berkley DB 概要, http://www.oracle.com/jp/database/berkeley-db/overview/index.html (参照 2017/6/1).

[13] Jongsung Won, Ghang Lee: Algorithm for Efficiently Extracting IFC Building Elements

from an IFC Building Model, 2011.

[14] G. Arthaud, J. C. Lombardo: Automatic Semantic Comparison of STEP Product Models, Innovations in Design & Decision support systems in Architecture and Urban Planning, pp.447–463, 2006.

[15] MongoDB, Inc.: What is MongoDB?, https://www.mongodb.com/what-is-mongodb (参照 2017/6/1).

[16] 川谷卓哉, 伊東栄典: プライベートクラウドを利用したセンサデータ蓄積基盤の試作, 情報処理学会研究報告, 2014.

[17] 桑野章弘: CyberAgent における MongoDB, MongoDB Tokyo 2013, 2013, https://thinkit.co.jp/story/2013/12/27/4744?page=0%2C1 (参照 2017/6/28).

[18] Neo Technology, Inc.: Neo4j: The World's Leading Graph Database, https://neo4j.com/product/ (参照 2017/6/1).

[19] Neo Technology, Inc: Exploring LinkedIn in Neo4j, 2013, https://neo4j.com/blog/exploring-linkedin-in-neo4j/ (参照 2017/6/28).

[20] Neo Technology, Inc: Neo4j Decreases Development Time-to-Market for LinkedIn's Chitu App, 2016, https://neo4j.com/news/neo4j-decreases-development-time-market-linkedins-chitu-app/ (参照 2017/6/28).

[21] Object Management Group, 西原裕善（訳）: UML2.0 仕様書 2.1 対応, オーム社, 2006.

[22] Martin Fowler, 羽生田栄一（監訳）: UML モデリングのエッセンス 第 3 版, 翔泳社, 2006.

第 2 章

IFC スキーマとプロダクトモデル

2.1　本章の概要

第 1 章において，BIM/CIM の当面の目標は，BIM/CIM 情報共有システムを構築し，多数の関係者間における情報共有を実現することである，と述べた．BIM/CIM で標準的に用いられている IFC スキーマは，オブジェクト指向分析設計 (OOAD: Object-Oriented Analysis and Design) 手法に基いて設計されていると考えられるが，オブジェクト指向分析設計が一般的に適用され，成果が上がっている事例（主に，ソフトウェア開発や，企業における業務分析）とは多くの相違がある．

本章では，IFC スキーマ開発の経緯や特徴を述べ，さらに，IFC スキーマによって記述されたプロダクトモデルの具体例を示す．それらにより，BIM/CIM 情報共有システムが備えるべき機能と IFC プロダクトモデルを対象とするデータ処理の問題点を明確にする．

2.2　IFC スキーマの成り立ち

前章で述べたように，BIM/CIM では建築物の 3 次元プロダクトモデルを IFC スキーマを用いて記述するのが一般的である．IFC は，国際的な標準化団体である buildingSMART International [1] の活動により，2013 年に国際標準 ISO16739 [2] になっており，多くの 3 次元 CAD が Part21 ファイルフォーマット [3] による IFC プロダクトモデルの入出力をサポートしている．Part21 ファイルフォーマットは IFC プロダクトモデルを完全に記述できるため，IFC スキーマでプロダクトモデルを記述することにより，複数のシステム（現状では主に 3 次元 CAD）間でプロダクトモデルを交換することが可能となっている．

IFC スキーマは，図 2.1 に示すように，順次更新されながら開発が進められたため，現時点では複数の異なるバージョンのスキーマが存在している．実用化以前のバージョンが使われることはほぼないが，デファクトスタンダードである IFC2x3 [4]，最新バージョンである IFC4 [5] でモデル化されたプロダクトモデルは，両者共に流通している．さらに，今後，新たなバージョンが完成すれば，そのバージョンでモデル化されたプロダクトモデルも流通することになるはずである．

図 2.2 に，2016 年に buildingSMART Standards Summit [6] で公表された IFC の将来構想を

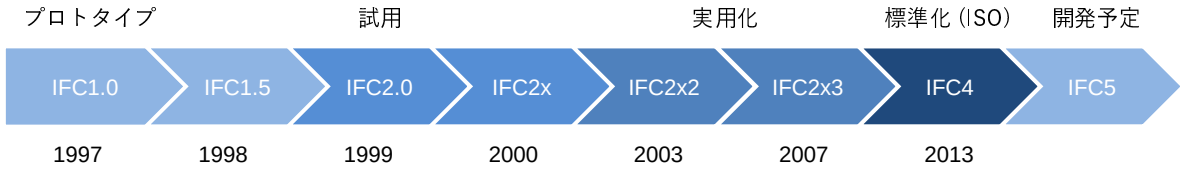

図 2.1　IFC 開発の歴史

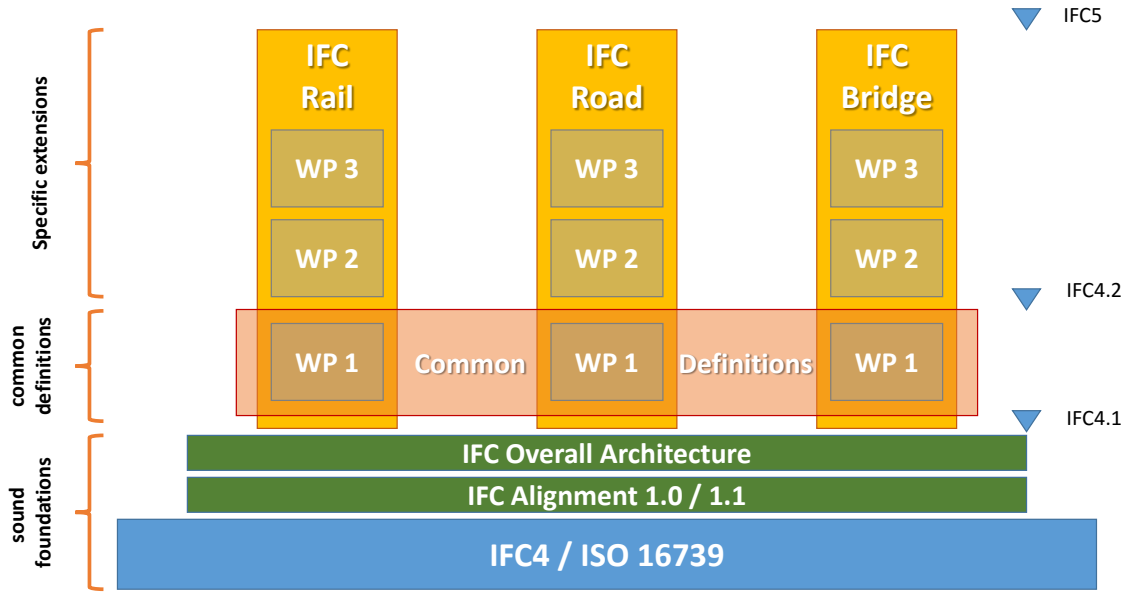

図 2.2　IFC の将来構想（出典: InfraRoom Resolutions bSI autumn summit 2016 Jeju）

示す．IFC4 では，主にビルや住居等の建築物をモデリングの対象としていたが，今後開発されるバージョンでは線形情報を含める予定となっており，さらに将来は，鉄道，道路，橋梁といった土木構造物も対象に含まれる予定となっている．

2.3　IFC スキーマの特徴

2.3.1　オブジェクト指向

IFC スキーマは，汎用的なデータ仕様記述言語 (Formal Data Specification Language) である EXPRESS [7,8] により記述されている．EXPRESS 言語では，記述対象とする物体や概念をエンティティと呼んでおり，各エンティティ毎に属性を定義できる．エンティティが実体化し，属性に具体的な値が与えられたものをインスタンスと言う．また，エンティティ間に継承[*1]関係を定義す

[*1] 継承先のエンティティが継承元の定義を引き継ぐこと．

ることや，制約[*2]を定義することが可能である．IFC スキーマでは，BIM の対象である建築物を構成する壁，ドア等のあらゆる物の抽象概念がクラスとして定義されている。なお，このエンティティという用語であるが，OOAD におけるクラスと同概念であり，クラスの方が一般的によく使われているため，以下では，クラスと表記することにする．

オブジェクト指向は，オブジェクト指向プログラミング言語 (OOPL: Object Oriented Programming Language) やオブジェクト指向データベース等様々に応用されているが，これらは，同様にオブジェクト指向に基づいている IFC スキーマと親和性が高いため，BIM/CIM 情報共有システムを構築する際に，これらを用いれば，比較的容易にシステム化できる可能性が高いと考えられる．しかしながら，EXPRESS とこれら言語やデータベースには，データ型や機能に相違があるため，その点に留意が必要になる．

例えば，オブジェクト指向モデリングでは，モデル化対象の静的な性質（属性）と動的な性質（振る舞い）の両面から，モデルを定義していく．しかしながら，EXPRESS にはモデルの振る舞いを定義する機能が存在しない[*3]．また，表 2.1–2.3 に示すように，継承の方法，属性の宣言方法，データ型に一般的な OOPL と相違がある．例えば，広く使われている OOPL である Java 言語 [10] を例に取って比較すると，以下の相違が見られる．

1. Java では，上位型であることを明示的に宣言する，つまり，自クラスの下位型を明示的に制限することはできない．
2. Java では，表 2.2 の意味を付与して，属性を宣言することはできない．
3. Java では，NUMBER, LOGICAL, DEFINED, SELECT に相当するデータ型は標準では提供されていない．

しかしながら，システム開発を想定した場合は 1 が問題になることはない．なぜなら，IFC スキーマに新たにクラスを定義することはないからである．また，2 と 3 については，Java の言語仕様には存在しないが，プログラムコードを記述することによって同等の機能を実現することは可能である．前述したように，EXPRESS はクラスの振る舞いを定義することができないため，OOPL では必要とされないような，非常に詳細な属性の定義方法，及びデータ型が用意されていると考えられる．

表 2.1　EXPRESS におけるクラスの宣言子

EXPRESSの宣言子	意味
SUPERTYPE	継承関係における上位型であることを宣言する．
SUBTYPE	継承関係における下位型であることを宣言する．
ABSTRACT SUPERTYPE	継承関係における上位型であり，かつ抽象クラスであることを宣言する．

[*2] インスタンスが取り得る状態を制限するための規則．

[*3] 過去には，OOPL におけるメソッドに相当する機能を拡張する計画 [9] があったようだが，実現していない．

表 2.2 EXPRESS における属性の宣言子

EXPRESSの宣言子	意味
DERIVE	属性値が，他の属性の値から導出される．
INVERSE	関連するエンティティ同士の逆方向の関連を明示する．
OPTIONAL	属性値の設定が任意である．
UNIQUE	属性値が一意である．
WHERE	属性値に制約を付与する．

表 2.3 EXPRESS におけるデータ型

EXPRESSのデータ型		意味
単純データ型	数値 (NUMBER)	あらゆる数値
	実数 (REAL)	実数
	整数 (INTEGER)	整数
	文字列 (STRING)	文字列
	論理型 (LOGICAL)	真，偽，不明のいずれか
	ブール (BOOLEAN)	真，または偽
	2進 (BINARY)	ビット列
集合体データ型	配列 (ARRAY)	固定長の順序付けられた集まり
	リスト (LIST)	位置によりアクセスできる要素の並び
	多重重合 (BAG)	重複が許された順序付けられていない集合
	集合 (SET)	重複が許されない順序付けられていない集合
名前付きデータ型	エンティティ (ENTITY)	属性等を宣言により定義した型
	定義 (DEFINED)	既存データ型の定義域を制約する等して新たに定義した型
構成データ型	列挙 (ENUMERATION)	名前の順序付けられた集まりを定義域とする
	選択 (SELECT)	複数の名前付きデータ型の定義域の和を定義域とする

2.3.2 関係性

IFC スキーマではインスタンス同士の関係性を示すためのクラスが多数定義されており，対象物の構成要素同士の関係性を詳細に記述することが可能となっている．なお，各クラス名は “Ifc” で始まり，関係 (Relationship) を示すクラスの場合はその次に “Rel” が付けられ，その次に関係が記される．また，単語間にスペースは設けず，各単語の頭文字を大文字にする．また，関連元を示す属性名は “Relating” で始まり，関連先を示す属性名は “Related” で始まる命名規則となっている．それらクラスの構成を UML [11] クラス図として記述したものを図 2.3 に示す．また，例として，ある建築物の 1 階フロア (IfcBuildingStorey) の壁 (IfcWall) と柱 (IfcColumn) がコンクリートでできている様子を図 2.4 に示す．

図 2.4 のように関連クラスを介在させて，インスタンス間の関係性を詳細に表現する手法は，

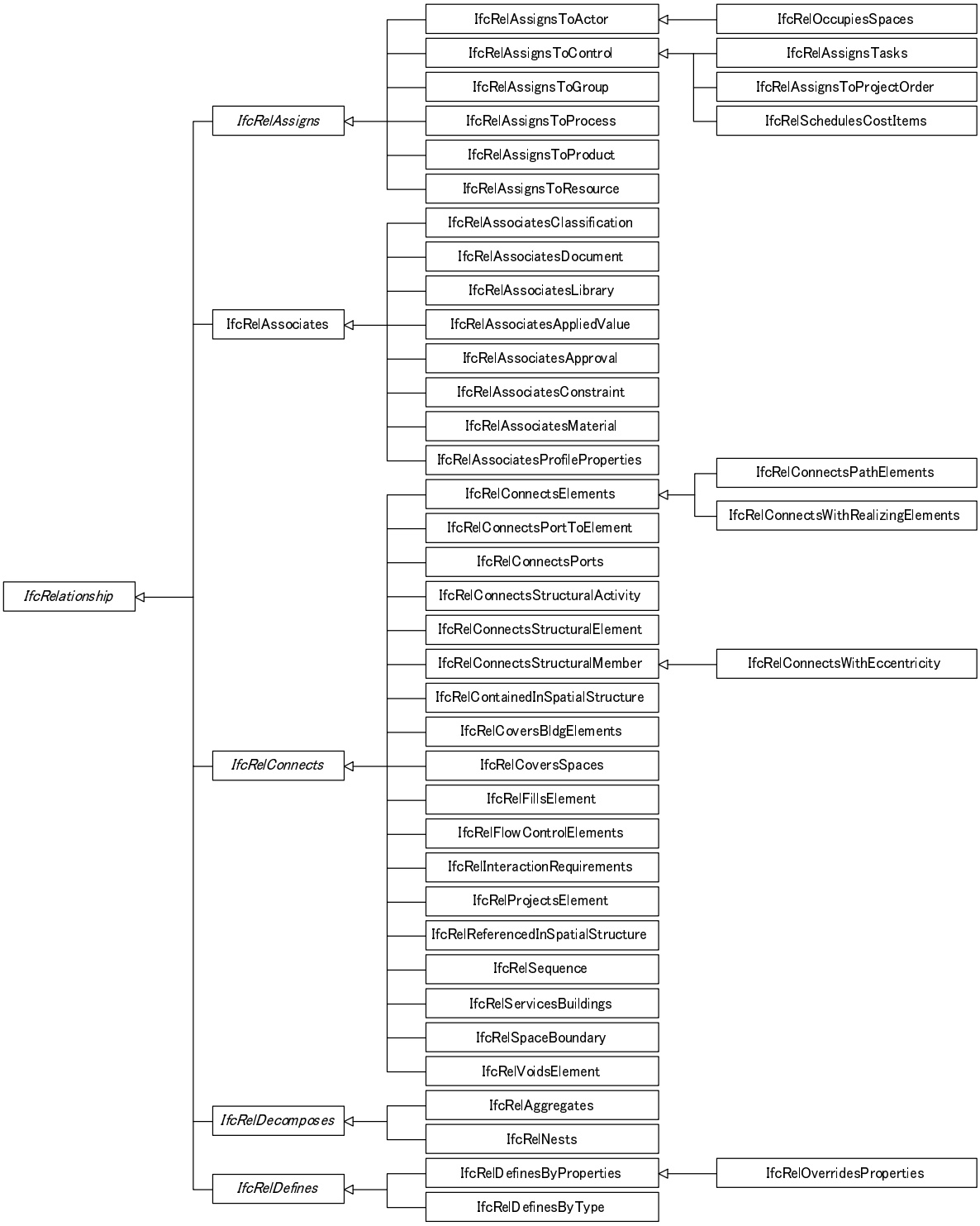

図 2.3　関係性を表わすクラスの構成（UML クラス図）

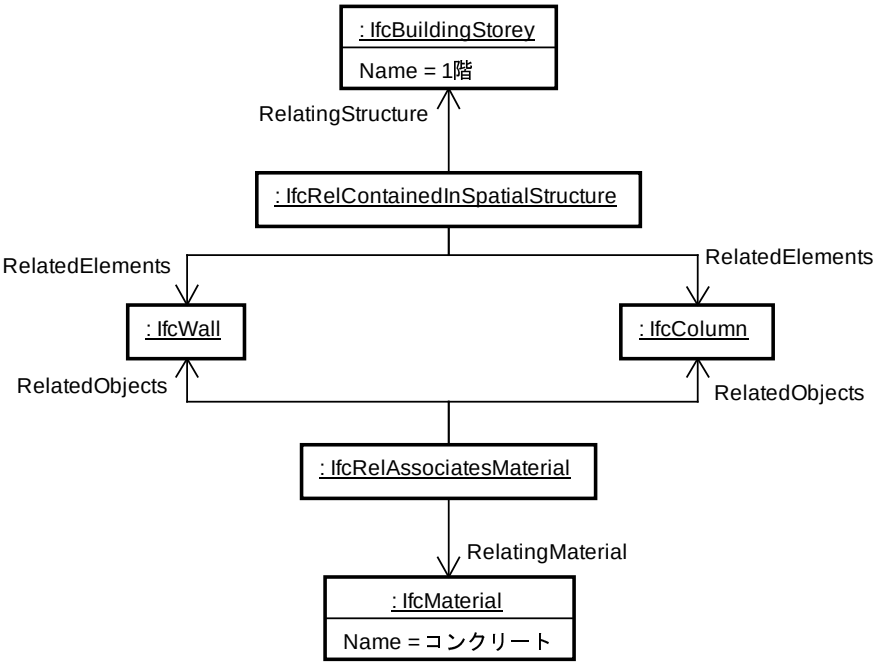

図 2.4 IFC における要素間の関係性の表現（UML オブジェクト図）

OOAD においても，しばしば用いられる一般的な手法であるが，図 2.4 の例では，単に，壁と柱に材料が紐付いているということを表現しているに過ぎず，関連クラスを設けてモデルを複雑にするほどのメリットがないように見える（その他の関係性を示すクラスにおいても同様である）．IFC スキーマのリファレンス [5] には，その辺の設計意図が記されていないが，プロダクトモデルをマシンリーダブルにすることを意図していると考えられる．

つまり，関連クラス（図 2.4 における IfcRelContainedInSpatialStructure と IfcRelAssociatesMaterial）により明示しなくとも，人間は，フロアと部材，部材と材料の関係性が異なることを理解できるが，コンピュータがそれらの違いを認識することは困難である．関連クラスを介在させることで，プロダクトモデルが複雑になるというデメリットは発生するが，プロダクトモデル内の関係性をソフトウェアが自動的に認識し，処理させることが容易になる．

2.3.3 拡張性

IFC スキーマは，ビルや住居等の建築物の 3 次元プロダクトモデルを，相互に交換するために，多くの種類の建築物を表現できるように汎用的に設計されている．つまり，特定の建築物が表現できれば良いだけでなく，あらゆる種類の構造，部材，材料を表現することが求められる．しかしながら，現実的には不可能であるため，IFC スキーマに拡張性を備えることで，この問題を解決している．以下に主な拡張性の仕組みを説明する．

(1) プロパティセット

IfcPropertySet クラスを用いると，クラス，またはインスタンスが保持できる情報の種類を動的に拡張できる．モデル化の際に，IFC スキーマで定義されている属性だけでは不足する場合は，IfcPropertySet を用いてプロパティを追加する．このようにして拡張したプロパティのうち，関連するものを一まとめにしてプロパティセットと呼んでいる[*4]．

IfcPropertySet クラスとそれに関連するクラスを図 2.5 に示す[*5]．IFC スキーマには様々な種類のプロパティが定義されているが，図 2.5 では，代表的な IfcPropertySingleValue, IfcPropertyListValue, IfcComplexProperty のみ図示している．IfcPropertySingleValue は 1 つの値のみを保持できるプロパティで，IfcPropertyListValue は値の順序付けられた集合を保持できるプロパティである．IfcComplexProperty は，これらプロパティの集合を保持することができるため，ツリー構造のような複雑な構造の情報も保持することが可能である．

実際にプロパティセットを用いて拡張する方法は 2 種類あり，ある 1 つのクラスを元に生成されたすべてのインスタンスが，同じ値のプロパティを保持する場合は，IfcTypeObject に関連付いたプロパティセットが IfcRelDefinedByType で IfcObject に関係付ける．一方，インスタンス毎に異なるプロパティ値を保持する場合は，IfcRelDefinedByProperties で IfcObject と IfcPropertySet を関係付ける．この 2 種類の方法を併用することも可能である．なお，IFC4 では 408 種類のプロパティセットがすでに定義されており，これらを用いることも，新たに定義して用いることもできる．

(2) プロキシクラス

モデリングの際に，IFC スキーマで定義されていない概念や要素が必要になったとする．IFC スキーマ自体を拡張することは可能であるが，周辺のアプリケーションを対応させるためには標準化が必要であり，それには非常に長い時間と多大な労力が必要になる．そこで，IFC スキーマには，そのような未定義の概念を代替するクラスが用意されている．それが，IfcProxy クラスと IfcBuildingElementProxy クラスである（図 2.6）．クラス図からも分かる通り，これらクラスには最低限の属性しか定義されていない．他に必要な属性が存在する場合は，前述したプロパティセットを用いて定義することになる．

IfcBuildingElementProxy は，IFC スキーマに定義されていない建築要素（例えば壁やドア）が必要になったときに用いられる．建築要素以外の概念が必要になった場合は，IfcProxy クラスを用いることになるが，IFC4 では DEPRECATION[*6]扱いとなっているため，事実上対処できない．

[*4] 本論では，EXPRESS で定義した静的な性質のことを属性 (attribute)，プロパティセットの仕組みを用いて動的に拡張した性質のことをプロパティと区別して呼ぶ．格納できる情報に大きな差異はない．

[*5] IFC のクラスを UML で記述する際は，集合体データ型の LIST を ordered 制約（要素が順序付けられているの意）で，SET を unique 制約（要素が一意であるの意）で記述した．また，属性の宣言子は UML で直接記述できないため，宣言子名をそのまま用い，ステレオタイプとして記述した．

[*6] “将来廃止する予定のクラスであり，互換性を保つ目的等の理由で残しているに過ぎないため，極力使用するべきではない” の意．

その場合は既存クラスから類似概念のクラスを選んで利用することになると考えられる．

2.4　IFC プロダクトモデル

前節で示した通り，IFC スキーマでは，プロダクトモデルを構成する要素（インスタンス）同士の間に，それらの関係性を表現するインスタンスが介在する設計となっている。また，IfcBuildingElementProxy クラスとプロパティセットを用いて未定義の要素を表現する場合は，定義済の要素を表現するのであれば 1 個のインスタンスで済む所を，図 2.5 からも分かる通り，多数のインスタンスを用いる必要が生じる．そのため，IFC プロダクトモデルは，深い階層構造を有した複雑なモデルになると考えられる。

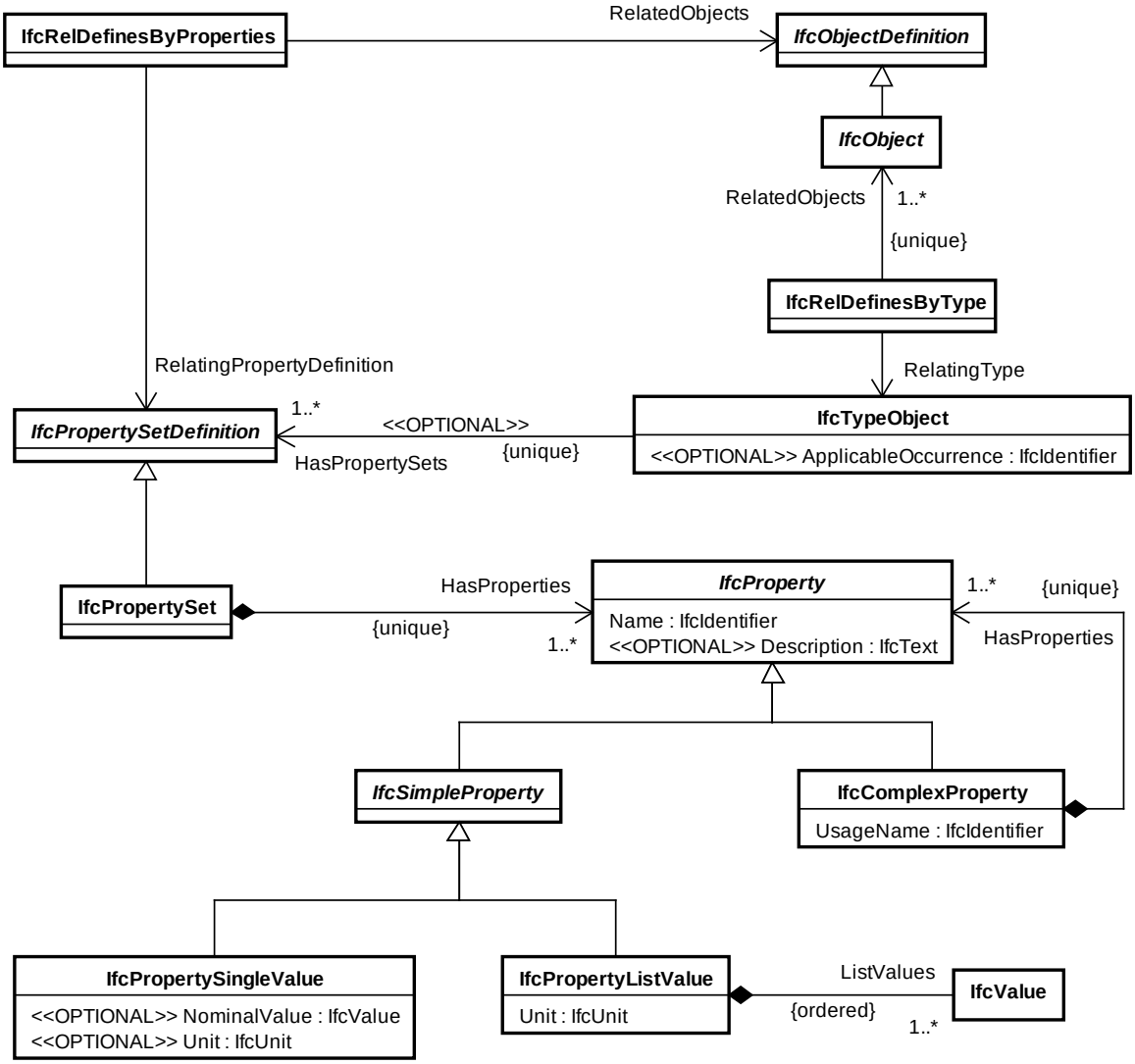

図 2.5　IfcPropertySet クラスとその関連クラス（UML クラス図）

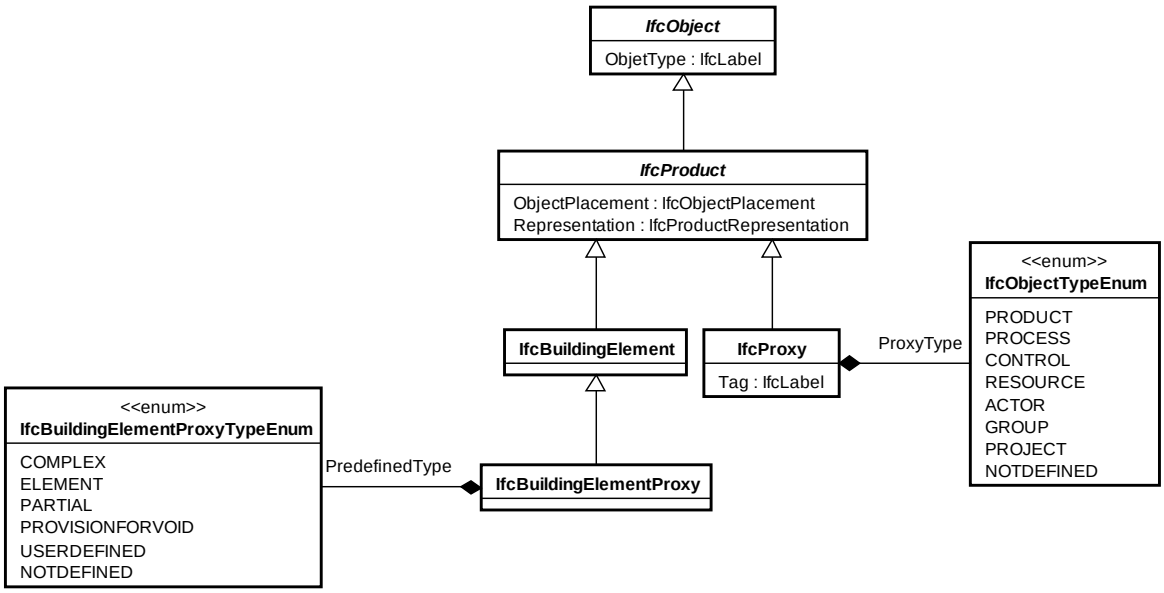

図 2.6　プロキシクラスと関連クラス（UML クラス図）

本節では，IFC プロダクトモデルの一部を例示し，IFC プロダクトモデルの複雑さを示す．

2.4.1　例 1: 5 階建てビルディング

5 階建てビルディングの IFC プロダクトモデルを UML オブジェクト図で記述したモデルの一部を図 2.7 に示す．網掛けの箇所が前述した関係性を表現するインスタンスである．この図から，ビルディング全体の空間 (IfcBuilding) が複数の各階毎の空間 (IfcBuilingStorey) に分割されており，各階の空間には物理的な構成要素である，階段 (IfcStair)，壁 (IfcWall)，ドア (IfcDoor) が含まれていることが分かる．これらは，1 対多の関係性を示す IfcRelAggregates，建築物の空間とそこに含まれる物理的な要素との関係性を示す IfcRelContainedSpatialStructure を用いて関連付けられている．また，ドア等の物理要素は，その材料と IfcAssociatedMaterial を介して関連している．

この例では，最上位の要素である IfcProject から末端の要素である IfcCartesianPoint（要素の座標値を示す）までの階層は 15 階層であり，その中に 4 つの IfcRelationship 派生インスタンスを含んでいる．これは，空間要素と物理要素が関連するという，プロダクトモデルが表現するべき本質的な構造のみを表わすのであれば 11 階層で済むが，関連クラスにより要素間の関係性を明示するという IFC スキーマの設計思想により，15 階層に増えたということを意味している．

2.4.2　例 2: 橋台

同様に，土木構造物である橋梁の下部構造の一部である橋台をモデリングする．図 2.8 に示すように，橋台は，ウイング，パラペット、堅壁等、複数の部材により構成されているが，これらの構

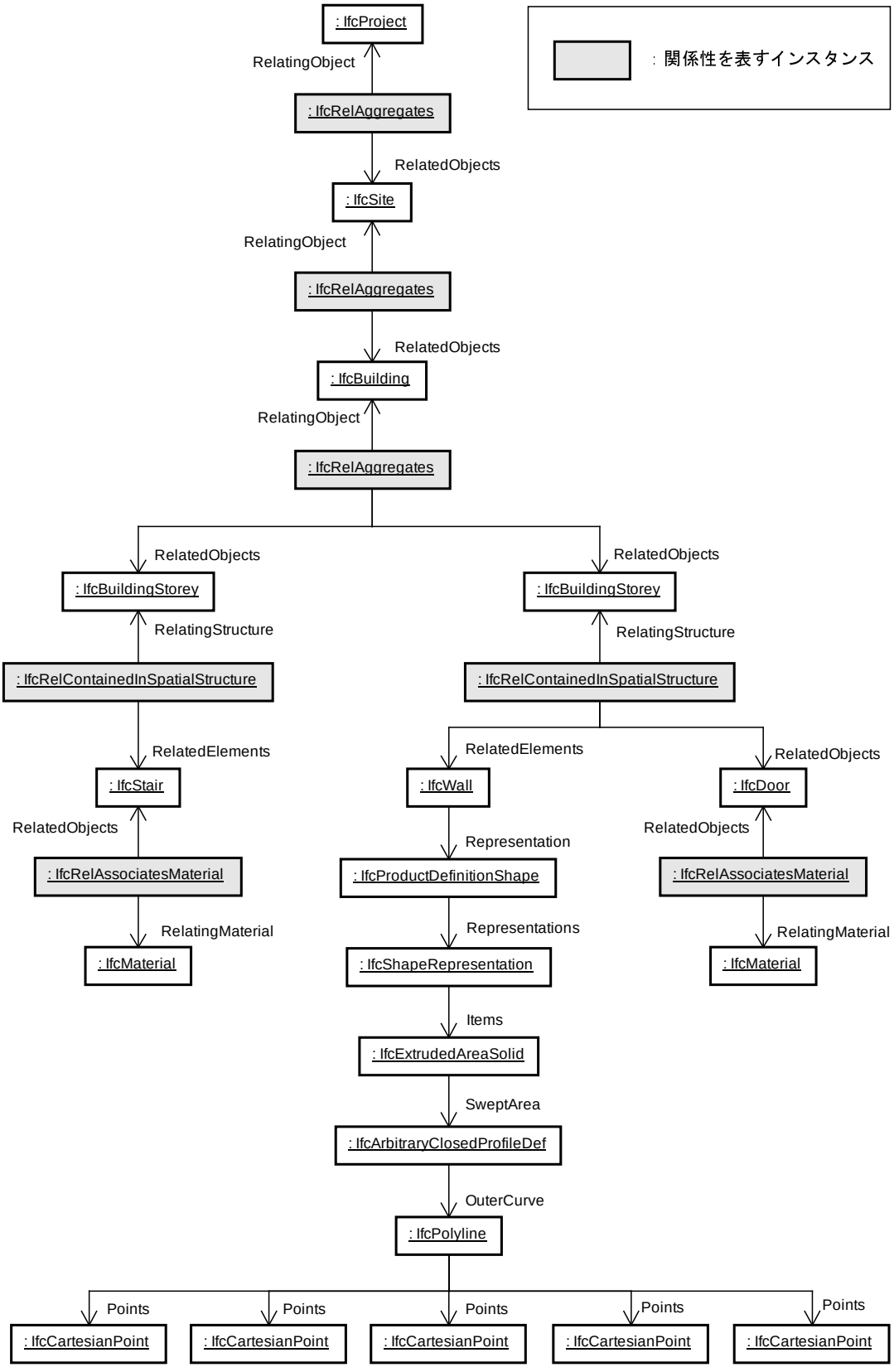

図 2.7　5F 建てビルディングのプロダクトモデルの一部（UML オブジェクト図）

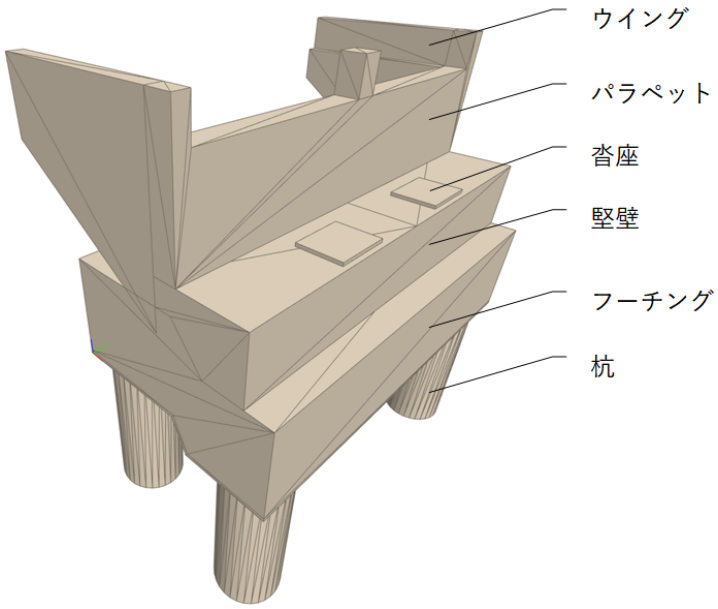

図 2.8　橋台の構造

造部材は，フーチング (IfcFooting) と杭 (IfcPile) を除いて，建築物を対象とする IFC スキーマには定義されていない．そのため，前述した代替用クラスである IfcBuildingElementProxy を用いてモデリングすることになる．また，橋梁全体，および橋台全体は空間要素としてモデリングしているが，当然，IFC スキーマにはこれらを表わすクラスが存在しないため，IfcBuilding（ビル全体の空間を表す）と IfcBuildingStorey（階の空間を表わす）で代替している．これは，2.3.3 項で述べたように，IfcBuildingElementProxy を空間要素として使えないためである．さらに，フーチングの配筋情報をプロパティセットにより表現している．

このようにしてモデリングしたプロダクトモデルの一部を図 2.9 に示す．網掛け箇所が，プロパティセットに関係するインスタンスである．図 2.9 ではフーチングのみにプロパティを関連付けているが，実際は他の構造部材にも配筋情報が必要であり，プロダクトモデルを構成するインスタンスの多数を占めることになる．

これは，IFC スキーマの拡張のための機能の 1 つであるプロパティセットを用いると，プロパティ 1 個につきインスタンスが 1 個必要になるため，プロダクトモデルが複雑になることを示している．プロパティセットは，この例で示したような，IFC が元来対象としていない物体をモデリングする際に用いられるだけでなく，既存概念（既存クラス）の拡張にも用いられる．現在は，IFC スキーマは形状表現のために用いられる場合がほとんどであるため，既存クラスを拡張するというケースはそれほど多くない．しかし，今後，BIM/CIM が成熟していくに連れ，IFC プロダクトモデルが様々に応用され，既存クラスに予め備わった属性だけでは不足し，プロパティセットを用いた拡張が多くなされるようになるとすれば，さらにプロダクトモデルが複雑になると考えられる．

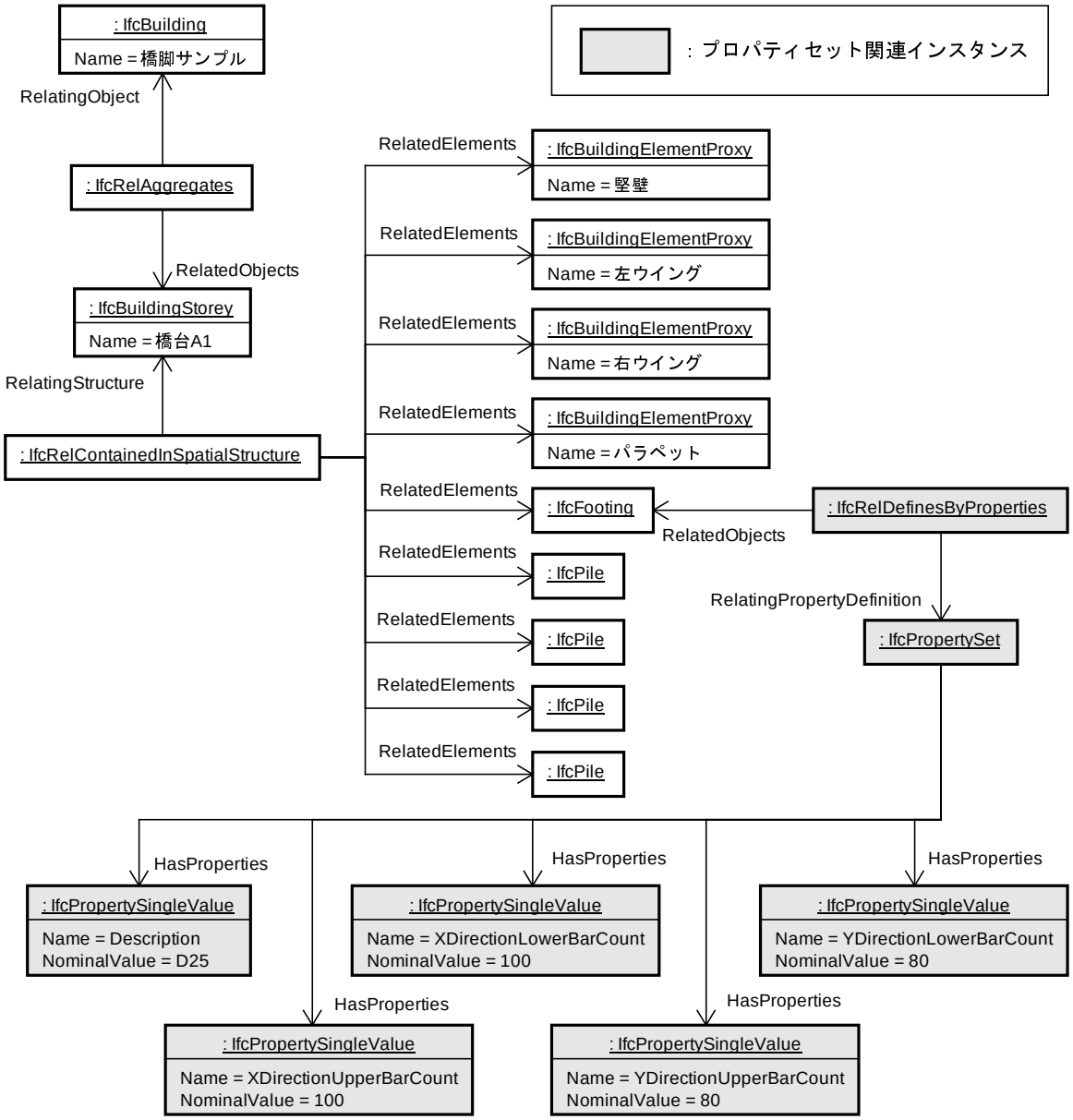

図 2.9　橋台のプロダクトモデルの一部（UML オブジェクト図）

2.5　CIM における IFC スキーマ

前述した通り，IFC スキーマはビルや住居等の建築物をモデリングの対象として設計された，BIM の実現を目的としたスキーマである．一方，CIM の実現を目的として，土木構造物を対象とする標準化されたスキーマは存在していない．これが，CIM における土木構造物のモデル化作業で大きな問題となっており，現状では，この問題には主に以下に示す 2 つの手法で対処されている．

2.5.1 土木構造物のモデリング手法

(1) IFC スキーマを用いた土木構造物のモデリング

一つは，建築物を対象とする IFC スキーマをそのまま用いて，モデリングする手法である．この場合は土木構造物特有の部材や概念は利用できないため，他の概念で代替する必要がある．橋梁を例に取ると，IFC スキーマには橋脚や橋桁を表わすクラスは定義されていないため，IFC スキーマに定義されている既存のクラスである IfcWall（壁を表わす）や IfcSlab（梁を表わす）等で代替してモデリングする．仮に，代替するための適当なクラスが存在しない場合は，前述した IfcBuildingElementProxy クラスを用いてモデリングすることになる．

この手法は，プロダクトモデルを交換する際に，互いに同じ規則で入出力する必要があるため，運用が難しくなるが，CAD 等，IFC スキーマに対応した既存システムをそのまま利用可能であるというメリットがあり，特定の組織やプロジェクト等，限定された範囲内でプロダクトモデルを交換できれば良い場合には有効である．この手法の代表的なものが，BrIM (Bridge Information Modeling) [12] であり，橋梁を既存の IFC スキーマでモデリングする際の規則をまとめたものである．

(2) IFC 拡張スキーマを用いた土木構造物のモデリング

もう一つの手法は，既存の IFC スキーマに必要なクラスを追加し，拡張した新たなスキーマ（以下では，IFC 拡張スキーマと言う）を作成し，それを用いてモデリングする手法である．土木構造物を対象として提案された IFC 拡張スキーマのうち，主なものを以下に示す．

- 橋梁: IFC-Bridge [13]
- 道路: IFC-Road
- 鉄道: IFC-Rail
- シールドトンネル: IFC-Shield [14]

これらは，対象となる土木構造物を想定して設計された専用のスキーマであるため，建築物を対象とする IFC スキーマをそのまま用いる手法に比べてモデリングしやすいメリットがあるが，標準化，および周辺ソフトウェアの対応に時間がかかる．現に，builidngSMART International が鋭意努力しているものの，これらは，まだ標準化には至っていない．

また，道路の盛土，切土，および地形データを記述するための，LandXML [15] という基準が存在するが，道路中心線，および横断面のモデリングに利用されているのみで，道路自体のモデリングには使われていない．また，名前が示す通り，XML (eXtensible Markup Language) 形式のスキーマであり，IFC をベースとしていない．

2.5.2 IFC 拡張スキーマへの対応

本研究では，CIM も含めた，IFC プロダクトモデルとデータベースの適合性を明らかにすることを主目的としているため，当然，IFC 拡張スキーマについても考慮する必要がある．図 2.2 で示した IFC の将来構想が実現し，建築物のスキーマと土木構造物のスキーマが IFC5 として統合された結果，IFC5 のみを考慮すれば良い状況になることが理想ではあるが，その実現にはまだ多くの時間がかかると予想され，しばらくは，土木構造物用の IFC 拡張スキーマが独立して存在する状況が続くと考えられる．

よって，本研究では，研究対象を標準化された IFC4 スキーマに限定することはせず，前バージョンである IFC2x3 や，IFC 拡張スキーマも含めたすべての IFC スキーマを研究対象とすることにする．つまり，IFC スキーマのデータ構造の本質に近い部分に着目して研究を行なうということである．

2.6 本章のまとめ

本章では，IFC スキーマとプロダクトモデルを様々な角度から分析し，以下の特徴が存在することを示した．

- IFC スキーマ，および IFC 拡張スキーマは，オブジェクト指向分析設計に基いて設計されており，IFC プロダクトモデルは，ツリー構造に近い複雑な階層を有する．
- IFC スキーマは，拡張性を考慮した設計とインスタンス間の関係性の表現を重視した設計により，さらに複雑な構造となる．
- BIM/CIM では（特に CIM において顕著である），IFC 拡張スキーマや異なるバージョンのスキーマ等，様々な種類のプロダクトモデルが流通する．
- BIM/CIM で用いる IFC プロダクトモデルは，対象となる建築物，土木構造物のライフサイクル全般に渡って，発生した情報を蓄積していくため，その情報量は膨大になる．

BIM/CIM において，多数の関係者間での情報共有を実現するため，BIM/CIM 情報共有システムを構築することを考えると，IFC プロダクトモデルを DBMS (DataBase Managemet System) に格納して管理する必要がある．前述したように，IFC プロダクトモデルは Part21 というテキストフォーマットで完全に記述することができるが，ファイルベースの共有では，アクセス速度や，排他制御等で問題が発生するからである．DBMS には様々な種類が存在するが，それぞれ対象とするデータモデル（データ構造）が決まっており，性能はそのデータモデルに特化している．つまり，システム化の際には，IFC プロダクトモデルのデータ構造に合わせて DBMS を選択する必要がある．

つまり，BIM/CIM 情報共有システムの構築の際には，上記の IFC プロダクトモデルの特徴に合わせて DBMS を選択し，さらに，複数種類，かつ複数バージョンの IFC スキーマに対応したシ

ステムを開発する必要がある.

参考文献

[1] buildingSMART International: About buildingSMART, http://buildingsmart.org/about/about-buildingsmart/ (参照 2017/6/1).

[2] ISO 16739:2013: Industry Foundation Classes (IFC) for data sharing in the construction and facility management industries, 2013.

[3] ISO 10303-21:2002 : Industrial automation systems and integration – Product data representation and exchange – Part 21: Implementation methods: Clear text encoding of the exchange structure, 2002.

[4] buildingSMART International: Industry Foundation Classes IFC2x. Edition 3 Technical Corrigendum 1, 2007, http://www.buildingsmart-tech.org/ifc/IFC2x3/TC1/html/ (参照 2017/6/1).

[5] buildingSMART International: Industry Foundation Classes IFC4 Official Release, 2013, http://www.buildingsmart-tech.org/ifc/IFC4/final/html/ (参照 2017/6/1).

[6] buildingSMART International: Standards Summit-Jeju, Korea, 2016, http://buildingsmart.org/event/standards-summit-jeju-korea/ (参照 2017/6/1).

[7] ISO 10303-11:2004: Industrial automation systems and integration – Product data representation and exchange – Part 11: Description methods: The EXPRESS language reference manual, 2004.

[8] JIS B 3700-11:2002: 産業オートメーションシステム及びその統合－製品データの表現及び交換－第 11 部：記述法：EXPRESS 言語，2002.

[9] 田中文基, 菊地慶仁: 形式的データ仕様記述言語 EXPRESS, JSPE-59-12, 1993.

[10] James Gosling, Bill Joy, Guy Steele, Gilad Bracha, Alex Buckley: The Java®Language Specification Java SE 8 Edition, 2015.

[11] Object Management Group, 西原裕善（訳）: UML2.0 仕様書 2.1 対応, オーム社, 2006.

[12] U.S.Department of Transportation Federal Highway Administration: Bridge Information Model Standardization VOLUME I–III, 2016.

[13] E. Lebegue, B. Fies, J. Gual, G. Arthaud, T. Liebich, N. Yabuki: IFC-BRIDGE V3 Data Model - IFC4 Edition R3, 2013.

[14] 矢吹信喜，東谷雄一朗，秋山実，河内康，宮亨：シールドトンネルのプロダクトモデルの開発

に関する基礎的研究，土木情報利用技術論文集，Vol.16，pp.261–268，2007.

[15] LandXML.org: Welcome Land Development Professionals!, http://www.landxml.org/ (参照 2017/6/1).

第 3 章

IFC スキーマとデータベース

3.1　本章の概要

データベースとは，多数の情報を格納し，格納データの検索，追加，更新，削除といったデータ操作の手段を提供するものである．現在では，様々な種類のデータベースが提案され，さらに多数の DBMS 製品が開発，実用化されている．通常は，データベースに格納できるデータの構造がデータベースの方式毎に定まっており，そのデータ構造をデータモデルと呼んでいる．データベースに情報を格納するには，格納対象となる情報をそのデータモデルに変換する必要があるが，データベースに格納される情報には，様々な種類のものが存在するため，中には変換が困難であるものや，変換したとしても，検索等のデータ処理性能が著しく低下してしまう種類の情報も存在する．あらゆる構造のデータを効率良く処理できる万能なデータベース方式は存在しないため，格納するデータの構造に合わせて，適切なデータベースを選択する必要がある．

本研究では，プロダクトモデルとデータベースの適合性を明らかにする過程で，様々な方式のデータベースを検討するため，本章でそれらデータベースの特徴，主にデータモデルに関する特徴を整理しておく．また，本研究では BIM/CIM 情報共有システムとして実際にシステム化することも想定しており，第 6 章でシステム化する際のシステム構成や開発コストについても論じるため，システム開発における問題点やデータベースの分散処理に関する特徴についても同様に整理する．

3.2　リレーショナルデータベース

1970 年に Codd によって提案された関係モデル [1] に基づいたデータベースであり，1980 年代から普及し始め，現在では様々な分野で広く用いられている．技術的に成熟しており信頼性が高く，運用に必要なツールが揃っていることがその理由である．

3.2.1　関係モデル

関係モデルでは，データを関係という表に似た構造で管理している．1 つの関係には複数の属性を含めることができる．各属性の値の集合が組であり，複数の関係に，和，差，交差，直積，選択，射影，結合，商といった関係代数演算を施すことができる．概念図を図 3.1 に示す．図 3.1 では，属性 1 から 属性 m までの m 個 の属性が定義された関係に，n 個 の組が格納されている様子を表している．値 m, n は，n 番目 の組の 属性 m の値を示す．

なお，関係モデルにおける関係，属性，組という用語は，データベース以外の分野でもよく用いられる一般的な単語であり，混同を避けるためか，代わりにテーブル，カラム，レコードという用語が用いられることも多い．本論においても，それにならって，テーブル，カラム，レコードを用いることにする．

3.2.2　正規化

関係モデルでは，冗長性が少なくなるようにデータを正規形にして格納することが前提とされており，データは複数のテーブルに分割されて格納される．正規形の種類は第 1–5 正規形とボイス・コッド正規形の 6 種類が存在し，テーブルをこれら正規形にすることを正規化と言う [2].

正規化が進むほど，データは複数のテーブルに分割されることになる．

3.2.3　リレーショナルデータベースを用いたシステム開発における問題点

本研究では，実際にシステム化することも想定しているため，リレーショナル DBMS を用いたシステム開発における問題点を以下に示す．

(1)　固定スキーマ

リレーショナルデータベースでは，格納するデータの構造を，データ型も含めて予め決めておく必要がある．この格納データの構造をスキーマと言い，リレーショナルデータベースのスキーマのように，格納データの構造を動的に変更できないスキーマを，固定スキーマと言う．リレーショナル DBMS においては，スキーマはテーブルの定義そのものである．図 3.2 に，SQL [3] によるテー

属性 (カラム) $_1$	属性$_2$	…	属性$_m$
値$_{1,\,1}$	値$_{2,\,1}$	…	値$_{m,\,1}$
値$_{1,\,2}$	値$_{2,\,2}$	…	値$_{m,\,2}$
…	…	…	…
値$_{1,\,n}$	値$_{2,\,n}$	…	値$_{m,\,n}$

関係（テーブル）

組（レコード）

図 3.1　関係モデル概念図

```
1  CREATE TABLE IfcProject(
2      GlobalId CHAR(22) NOT NULL,
3      Name VARCHAR(255) NULL,
4      Description VARCHAR(255) NULL,
5      ObjectType VARCHAR(255) NULL,
6      LongName VARCHAR(255) NULL,
7      Phase VARCHAR(255) NULL,
8      OwnerHistoryId INTEGER NULL,
9      UnitAssignmentId INTEGER NULL
10 )
```

図 3.2 SQL によるテーブル定義例

ブル定義例を示す．1 行目でテーブル名，2–9 行目でカラム毎に，カラム名，データ型，NULL 値を許容するかどうかを定義している．

このように，格納されるデータが予め決まっており，定義した形式以外のデータを格納することができないため，システム化において以下のデメリットが存在すると考えられる．

1. 対象データの構造に変更があった場合は，必ずシステム改修が必要になる．
2. 多様な種類の構造のデータを対象とするシステムである場合は，システム構築費用が高額になる．
3. 対象データの構造が定まっていない，つまり，入力データが XML (eXtensible Markup Language) のような構造の自由度が高い半構造データ [4] である場合は，システム化自体が困難である．

第 2 章で，IFC スキーマの複数のバージョンや，IFC 拡張スキーマも含めた複数種類のスキーマに対応しなくてはならないと述べた．さらに，IFC プロダクトモデルの構造は階層が深く，複雑であるとも述べた．したがって，上に挙げた理由により，リレーショナルデータベースの利用は，BIM/CIM 情報共有システムを構築する際にデメリットになる可能性が高いと予想される．

(2) 結合操作の多発

3.2.2 項で説明したように，リレーショナルデータベースでは，正規化によりデータが複数のテーブルに分割される．分割されたデータを取得する際は，各テーブルのデータを組み合わせる必要が生じる．この演算を結合と言う．格納対象データの構造が複雑であるほど，多数の結合演算が発生するため，システム化の際には演算速度の面で留意が必要になる．

結合演算に要する計算量は，n 個のテーブルを結合するとすれば，各テーブルに格納されているレコード数をそれぞれ $N_1, N_2, ..., N_n$ として，以下の式で表わせる[*1]．

$$O(\log_2 N_1 + \log_2 N_2 + ... + \log_2 N_n)$$

[*1] DBMS の内部データ構造（内部スキーマ）が二分木である場合に限る．通常は，リレーショナル DBMS の内部スキーマは二分木であるが，ハッシュ構造等，他のデータ構造をオプションとして選択できる DBMS も存在する．

テーブル 1 個に対する計算量は $O(\log_2 N_n)$ であり十分高速であるが，計算量がテーブル数に比例するため，結合対象のテーブルが多くなると，データ取得処理で許容できない遅延が発生すると考えられる．

繰り返しになるが，IFC プロダクトモデルはデータ構造が複雑である．したがって，システム化の際には，結合演算が多発することが予想される．

(3) インピーダンスミスマッチ

昨今は，システム開発には，Java や C++ といった OOPL を用いるのが主流となっている．OOPL を用いると，ソフトウェアの再利用性と拡張性が向上するため，ソフトウェア開発の生産性が高まるのが，用いられている理由である．しかしながら，OOPL を用いたシステム開発でリレーショナル DBMS を用いると，しばしばインピーダンスミスマッチという問題が発生する [5]．以下に示すように，OOPL とリレーショナルデータベースには，主に 2 つのミスマッチが存在する．

1. データ構造のミスマッチ

   前述した通り，リレーショナル DBMS は関係モデルという 2 次元の表形式のデータ構造を扱う．一方，OOPL では複数のインスタンス同士が，一方向，または双方向に関連し合い，ネットワーク構造を構成する．両者のデータ構造に大きな相違があるため，アプリケーションレイヤーとデータベースレイヤー間でデータ授受が発生する度に，データ構造の変換が必要になる．このミスマッチは，変換処理実行によるシステム性能低下と，変換処理記述による開発コストの増加を招く[*2]．

2. 操作対象のミスマッチ

   OOPL では，常に 1 個のインスタンスを操作対象とし，集合を直接操作する方法は用意されていない[*3]．よって，オブジェクト指向プログラミング手法を用いて開発を行なうと，リレーショナル DBMS へのアクセスが頻繁に発生し，アプリケーションレイヤーとデータベースレイヤー間でのデータ授受がボトルネックとなる．これを嫌い，SQL によるデータ集合に対するアクセスを実行するようにプログラミングすると，オブジェクト指向プログラミングが崩れて生産性が落ちる．このミスマッチは，個であるインスタンス（オブジェクト）を操作対象とする OOPL と，集合を操作対象とする SQL の違いに端を発している．

[*2] 現在では，DBMS 自体の性能や，サーバ機器の性能が向上したため，変換処理実行によるオーバーヘッドはそれほど大きな問題とはなっておらず，開発コストの問題の方が重要視されている．

[*3] OOPL で集合を扱うには，配列等に格納された多数のインスタンスに対して，1 インスタンスずつ繰り返し操作を適用する．一方，SQL は，SELECT 句や UPDATE 句による，条件に合致するレコードの集合に対する演算を基本としている．

3.3 データベース開発の歴史

3.3.1 リレーショナル DBMS の普及

1970 年に Codd により関係モデルが提案されてからしばらく後，1980 年頃より，リレーショナル DBMS が製品化され始めた．1979 年に Oracle RDBMS，1981 年に IBM SQL/DS，1983 年に IBM DB2 が製品化され，1980 年代に普及期を迎えた．現在も広く用いられており，8 割のシェアを維持している [6].

リレーショナル DBMS が普及した主な理由を以下に挙げる．

1. 関係モデルという強固なデータベース理論が存在していたため，DBMS のベンダーは自ら理論を構築する必要がなく，ソフトウェアの開発に注力可能であったため，様々な開発ベンダーに支持された結果，多くのリレーショナル DBMS が製品化されて普及した．
2. すべてのリレーショナル DBMS が，同じデータモデルに基いているため，利用者（主にソフトウェア開発者）は異なる DBMS であってもノウハウを流用することが可能であり，利用者に支持されて普及した．
3. DBMS への問い合わせのための言語である SQL が 1986 年に米国国家規格協会 (ANSI : American National Standards Institute）で標準化されたため，利用者は異なる DBMS 製品に対しても統一的な手段で問い合わせ可能になり，利用者に支持されて普及した．
4. 関係モデルが比較的シンプルなデータモデルであったため，DBMS の性能を上げることが可能であり，利用者（主にエンドユーザー）に支持され普及した．

しかしながら，3.2.3 で示したような問題点もあり，それらが動機となって，リレーショナルデータベース以外のデータベース方式が研究，開発されることになったと考えられる．

3.3.2 分散データベースと NoSQL

NoSQL は，元々は “SQL を使わない”，つまりリレーショナルデータベース以外のデータベースの総称として使われていたが，現在は，“Not only SQL” の意味，つまり，“SQL だけではない” データベースということになっている．いずれにせよ，NoSQL に分類されるデータベースの本質を指した言葉ではない．実際に NoSQL に分類されているデータベースを見ると，それぞれのデータモデルが異なっており，同じ種類のデータベースには見えない．また，オブジェクト指向データベースや XML データベースは SQL を用いないにも関わらず，NoSQL には分類されていない．

“NoSQL” という用語は，非リレーショナル型 DBMS である Google BigTable [7] の登場がデータベース研究者，および開発者に与えた衝撃が大きく，非リレーショナル型の DBMS 開発がある種のトレンドになった結果，様々な方式の DBMS が登場し，それらを分類するための用語が必要になったため発明された用語に過ぎない．よって，それぞれのデータベースに本質的な共通点が見られないのは当然と言える．しかしながら，あえて，NoSQL データベースの共通点を挙げるとす

れば，BigTable の影響を受け，分散処理を意識して設計されだデータベースであるという点であると言える．これは，BigTable と共に有名になった CAP 定理 [8,9] を意識して設計されたデータベースと言い替えても良い．

(1) CAP 定理

CAP 定理とは，ウェブサービスのような分散コンピューティングシステムにおける定理であり，具体的には，以下の性質を同時に 2 つしか満たすことができないことを示す定理である．

1. 一貫性 (Consistency)
2. 可用性 (Availability)
3. 分断耐性 (Partition-tolerance)

ある分散コンピューティングシステムにおいて，データが複数のデータベースサーバに分散して格納されているとする．あるデータが更新された場合に，分散したデータが一度に更新されることはなく，更新データは時間をかけてシステム全体に伝搬していく（図 3.3）．すべてのデータが更新されるまでの時間はシステムにより異なるが，データベースサーバ同士が地理的に離れている場合や，分散度合いが高い場合は，より長くなると考えられる．

更新が伝搬している最中に，複数の参照リクエストがあった場合は，システムはそれぞれのリクエストに異なるレスポンスを返す可能性がある．これが，一貫性がない状態である．一貫性を満たそうとすると，更新データの伝搬が完了するまでデータアクセスをロックすることになり，利用者は待ち状態になる．これが，可用性が失なわれた状態である．一貫性と可用性を両立しようとすると，データを単一サーバに格納する以外にない．これが，分断耐性が失われた状態であり，そもそも分散型システムではなくなってしまう．

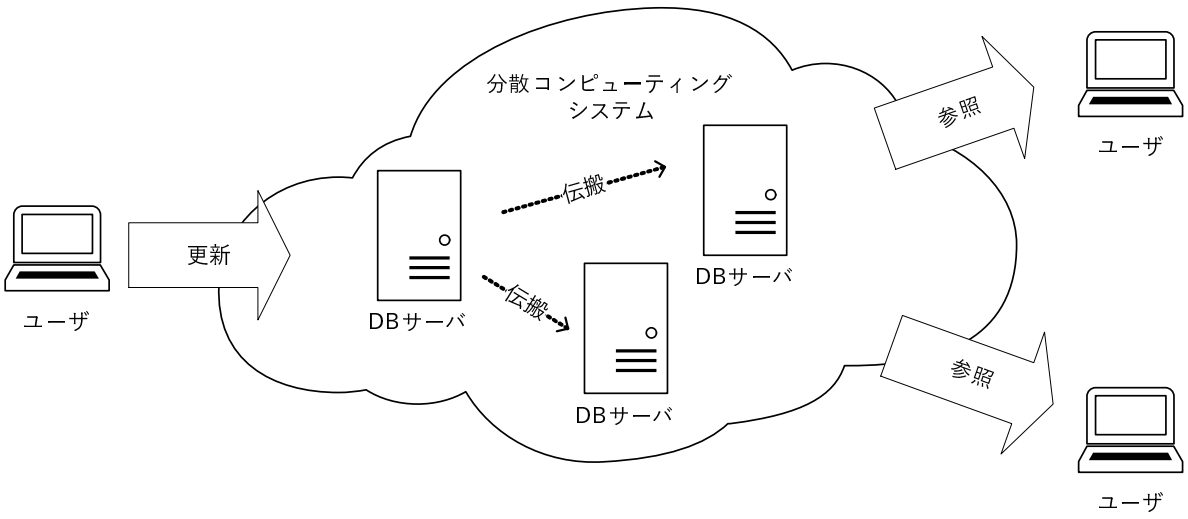

図 3.3 分散コンピューティングシステムにおける更新の伝搬

(2) 分散コンピューティング

CAP 定理をデータベースに当てはめると，リレーショナルデータベースは一貫性と可用性を求め，分断耐性を犠牲にしている．つまり，分散しない．一方，NoSQL データベースは，分断耐性を満たし，一貫性または可用性を犠牲にするような設計となっている場合が多い[*4]．分散可能であるということは，スケールアウト可能である，つまり，分散度合いを増すことでシステムの性能を向上させることが可能であるということであり，格納データ量，スループット（単位時間あたりの処理可能リクエスト数）の向上が期待できる．

3.4 非リレーショナル型データベース

リレーショナルデータベース以外のデータベース方式について主なものを以下に説明する．本論は，BIM/CIM 情報共有システムを実際にシステム化することを前提としているため，研究段階のものは対象とせず，十分な実績があり，長期の連続運用に耐え得る DBMS 製品が存在するデータベース方式に限定した．

3.4.1 オブジェクト指向データベース

オブジェクト指向技術に基いたデータベースである．当然，OOPL と適合性が高く，リレーショナルデータベースを用いたシステム開発で問題となっていたインピーダンスミスマッチが解消され，ソフトウェア開発の生産性が大きく向上するメリットがある．さらに，Java 等の OOPL と同程度の表現力を持っているため，格納対象となるモデルがかなり複雑なデータ構造であったとしても，難なく記述可能である．また，DBMS 製品にも依るが，スキーマレスである DBMS が多く，そうであれば，様々な形式のオブジェクトを格納可能であり，柔軟性が高いシステムが構築できると考えられる．

このように，リレーショナルデータベースを用いた開発の問題点を，ほぼすべて解消できるにも関わらず，十分なシェアを獲得するには至っていない．考えられる理由としては，データモデルが複雑過ぎるため，性能，機能，保守性を高いレベルでバランス良く実現するのが困難である点が挙げられる [10].

3.4.2 キー・バリューストア

キー・バリューストアと呼ばれる非常にシンプルな構造のデータベースである．キー（通常は文字列）に対応する値（文字列，またはバイナリデータ）を格納するだけである．概念図を図 3.4 に示す．図 3.4 上段では key1 から key4 までのキーがそれぞれ，A，B，C，D のバリューに対応し

[*4] 多くの NoSQL データベースが，可用性と分断耐性を満たす設計となっており，Google BigTable, Amazon SimpleDB, Apache Cassandra がこれにあたる．数は少ないが一貫性と分断耐性を満たすような DBMS もあり，Apache HBase がこれにあたる．

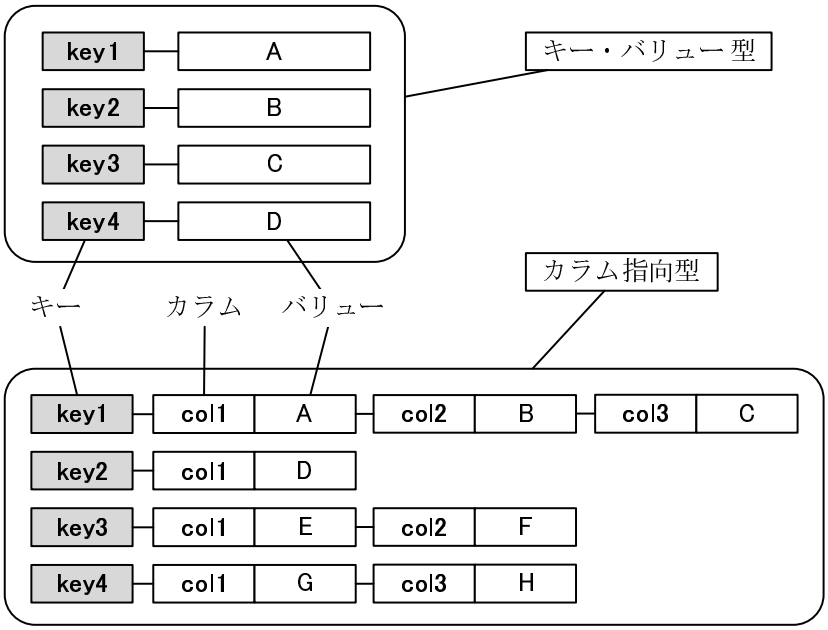

図 3.4 キー・バリューモデル概念図

て格納されており，キー・バリューのペア同士に何の関連もないことを表している．

3.4.3 カラム指向データベース

利便性を上げるため，1 つのキーに対応した複数のキー・バリューペアを格納できるようにしたカラム指向型と呼ばれる実装もあり，こうすると，表のようなデータ構造も表現できる（図 3.4 下段）．これら実装では，キー・バリューペアのデータ構造をバイナリデータに変換して 1 個のバリューとして格納しているだけで，内部のデータ構造は前述のシンプルなキー・バリューモデルと変わらない．

3.4.4 ドキュメント指向データベース

キー（通常は文字列）に対応するドキュメントと呼ばれる構造的データを管理可能なデータベースである．ドキュメントの形式（データモデル）に定義はないが，JSON 形式や XML 形式である DBMS が多い．

一般的に，同方式のデータベースは同じデータモデルを対象とするものだが，ドキュメント指向データベースは従来のデータベースが対象とするデータモデルよりも複雑で記述の制限が緩いデータ構造を対象にしたデータベースの総称であり，データモデルが製品に依存している．

1. JSON (JavaScript Object Notation) [11]
   JavaScript におけるオブジェクト構造の表記法をベースとしており，オブジェクト指向に基いたオブジェクト同士の関係を表現することができる．IFC もオブジェクト指向をベース

としており，データ構造の類似性は高い．

2. XML (eXtensible Markup Language) [12]
   データ表記法の一種で，IT システム間のデータ交換によく用いられている．階層化したツリー構造状のデータ構造を記述するのに適している．IFC プロダクトモデルもツリー構造であるため，データ構造の類似性は高い．

3.4.5 グラフデータベース

グラフ理論 [13] に基づいたデータベースであり，ノードとノード同士を結ぶ方向性を持つエッジにより構成されたグラフ（図 3.5）を管理することができる．汎用的なデータストアとして使えるように，各ノードとエッジはラベルとプロパティ（キーにより識別される値）を保持することができる．ラベルは各ノードとエッジ毎に 1 つ，プロパティは複数保持することができる．このデータモデルをプロパティグラフモデル [14] と言う．概念図を図 3.6 に示す．

プロパティグラフモデルは，ネットワーク状のデータ構造を表現するのに適している．IFC プロダクトモデルはほぼツリー状のデータ構造であるが，ツリー構造もネットワーク構造の一種である

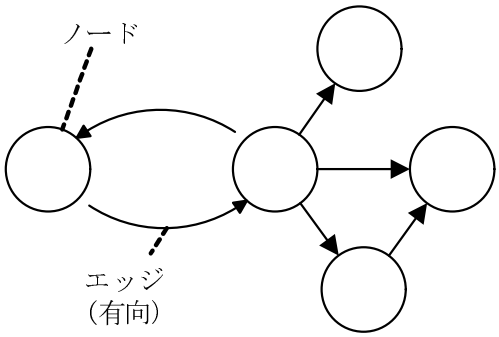

図 3.5　有向グラフ

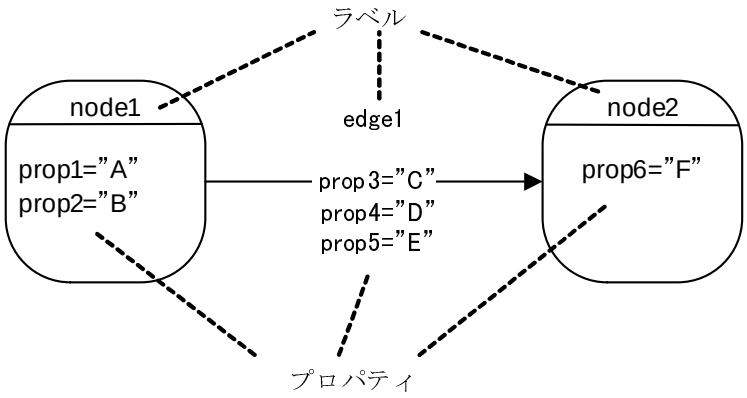

図 3.6　プロパティグラフモデル

ため，データ構造はある程度類似していると言える．

3.5 本章のまとめ

本章では，NoSQL に代表される非リレーショナル型のデータベースが，リレーショナルデータベースの弱点を補うようにして開発されてきた経緯を示し，さらに，各データベースの特性を，以下の視点から分析した．

- データモデル
- 分散コンピューティング
- システム開発

第 1 章，第 2 章で明らかにしたように，BIM/CIM で扱われる IFC プロダクトモデルは，データ量が膨大で，かつ深い階層構造を有しており，さらに，対象とする構造物のライフサイクル全般に渡って，数多くの関係者間で共有されなくてはならない．このような特殊なデータのマネジメントと利用環境を実現するために，上記の視点を含んだ多角的な視点からデータベースを検討，分析することにより，BIM/CIM 情報共有システムに適したデータベースを明らかにする必要がある．

参考文献

[1] E. F. Codd: A Relational Model of Data for Large Shared Data Banks, 1970.

[2] C. J. Date, 株式会社クイープ（訳）: データベース実践講義，オライリージャパン，2006.

[3] ISO/IEC 9075:1999: Database Language SQL, 1999.

[4] 田島敬史: 半構造データのためのデータモデルと操作言語, 情報処理学会論文誌 Vol.40 No.SIG3(TOD 1), 1999.

[5] TECHSCORE: Hibernate の基本, 2001, `http://www.techscore.com/tech/Java/Others/Hibernate/01/` (参照 2017/6/27).

[6] DB-ENGINES: DBMS popularity broken down by database model, 2017, `https://db-engines.com/en/ranking_categories` (参照 2017/6/27).

[7] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber: Bigtable: A Distributed Storage System for Structured Data, 7th USENIX Symposium on Operating Systems Design and Implementation, 2006.

[8] Eric A. Brewer: Towords Robust Distributed Systems, 2000.

[9] Seth Gilbert, Nancy Lynch: Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services, ACM SIGACT News, Volume 33 Issue 2, p51–59, 2002.

[10] 鶴岡邦俊, 木村裕: オブジェクト指向 DBMS のアーキテクチャに関する考察, データベースシステム, 1994.

[11] Ecma International: ECMAScript®2015 Language Specification, 2015, `http://www.ecma-international.org/ecma-262/6.0/index.html` (参照 2017/6/1).

[12] W3C: Extensible Markup Language (XML) 1.1 (Second Edition), 2006, `http://www.w3.org/TR/xml11/` (参照 2017/6/1).

[13] J. A. Bondy, U. S. R. Murty, 立花俊一（訳），奈良知恵（訳），田澤新成（訳）: グラフ理論への入門，共立出版，1991.

[14] Ian Robinson, Jim Webber, Emil Eifrem, 佐藤直生（監訳），木下哲也（訳）: グラフデータベース Neo4j によるグラフデータモデルとグラフデータベース入門，オライリージャパン，2015.

第 4 章

各種データベースによるプロダクトモデル管理手法の開発

4.1　本章の概要

IFC プロダクトモデルを DBMS に格納し，管理するためには，第 3 章で述べたように DBMS が採用するデータモデルに変換する必要がある．本章では，各種データベースの IFC プロダクトモデルへの適合性を明確にすることを目的として，IFC プロダクトモデルを各種データモデルに変換する手法を考案し，それら手法を用いて，様々な方式の DBMS に IFC プロダクトモデルを格納した上で，モデルの抽出処理速度を計測した．

4.2　データモデル変換手法

IFC スキーマを，第 3 章で示した各データベースのデータモデルに変換し，IFC プロダクトモデルを DBMS に格納することを考える．1 つのデータモデルに対する変換方法が複数考えられる場合もあるが，各データモデルの特徴を極力残したまま変換するようにした．本章では，IFC プロダクトモデルを DBMS で管理する際の，各データモデルそのものの適性を明らかにすることを目的としており，データ変換手法の優劣を比較することを目的としているわけではないからである．

説明のため，IFC プロダクトモデルの一部を各データモデルに変換する例を示すこととし，変換元プロダクトモデルを Part21 フォーマットで記述したものを図 4.1 に，UML オブジェクト図で記述したものを図 4.2 に示す．これは，IFC 拡張スキーマの一種である IFC-Bridge スキーマ [1,2] で記述した PC 箱桁橋のプロダクトモデルの一部である．

IFC-Bridge スキーマで定義されているクラスは，表 4.1 から分かるように，ほとんどのクラスは IFC4 のクラスを引き継いでいる．また，空間要素の記述方法が IFC4 とは異なっているが，第 2 章で示したような，プロダクトモデルのデータ構造上の特徴はそのまま引き継いでいる．そのため，本章で行なう検討は，IFC-Bridge だけでなく，IFC4 を含む IFC スキーマ全バージョン，おそらくは他の IFC 拡張スキーマにも適用できるはずである．

```
1 #1=IfcBridge('HXGKy49hQ7+r6h0f3Nt9pQ',$,'PC箱桁橋',.
  BOX_GIRDER_BRIDGE.,.COMPLEX.,.COMPOSITE.);
2 #2=IfcRelAggregates('7ET0zbEvSAC8onM3i9kebw',$,$,$
  ,#1,(#3,#4,#5,#6));
3 #3=IfcBridgePart('MHmkXG1OTNuirgG2T1i08g',$,'A1',$,.COMPOSITE.,.
  COMPLEX.,.PIER.,.MASSIVE_SECTION_ELEMENT.);
4 #4=IfcBridgePart('nmVdSHIzRWu/6h9L/b1/eA',$,'P1',$,.COMPOSITE.,.
  COMPLEX.,.PIER.,.MASSIVE_SECTION_ELEMENT.);
5 #5=IfcBridgePart('DQ9PeM7OQ4eoNsV9E+x+SA',$,'A2',$,.COMPOSITE.,.
  COMPLEX.,.PIER.,.MASSIVE_SECTION_ELEMENT.);
6 #6=IfcBridgePart('kEWTGkwERg2sOZiMJxwABw',$,$,$,.COMPOSITE.,.
  COMPLEX.,.DECK.,.UNICELLULAR_MONO_BOX-GIRDER.);
```

図 4.1　変換元 PC 箱桁橋プロダクトモデル（Part21 形式）

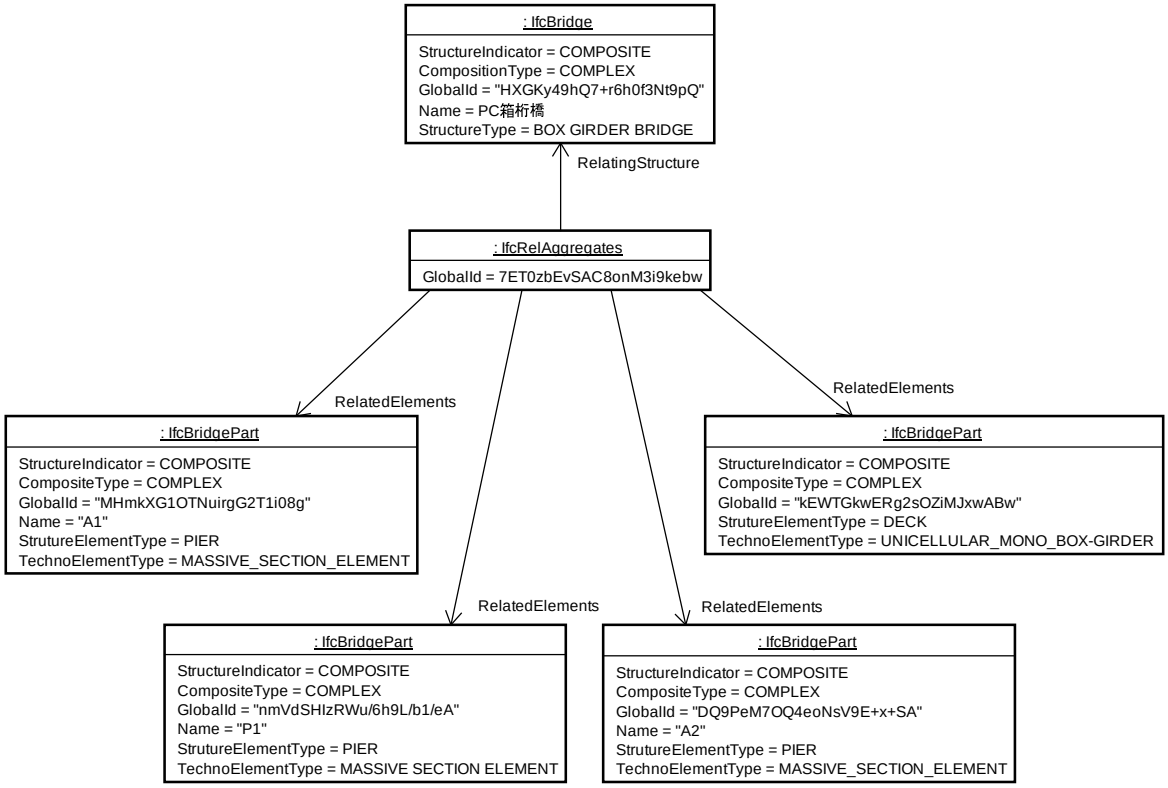

図 4.2　変換元 PC 箱桁橋プロダクトモデル（UML オブジェクト図）

表 4.1　IFC4 と IFC-Bridge スキーマのデータ型

データ型	IFC4	IFC-Bridge
エンティティ（ENTITY）	766	+21
定義（DEFINED）	126	0
列挙（ENUMERATION）	206	+9
選択（SELECT）	59	0

4.2.1 関係モデルへの変換

リレーショナルデータベースのデータモデルである関係モデルへの変換手法を考える．リレーショナルデータベースは正規化されたデータを格納することを前提としているため，IFC スキーマの正規化を試みる．完全に正規化されて一切の冗長性が排除されたデータベースは，多数のテーブル（関係）に分割されるためパフォーマンスが低下する場合がある．そのため，実用的には第 3 正規形が実現できた時点で十分に正規化されたと考えるのが一般的である．これにならって，IFC プロダクトモデルを関係モデルに変換する際も，第 3 正規形まで正規化することにする．

(1) 正規化

1. 第 1 正規化

   これは，繰り返しを排除するものである．繰り返しがあるデータを別のテーブルに分割すれば良い．IFC スキーマでは，集合体データ型の属性で 1 対多に関連しているクラス同士を別のテーブルとして定義する．

2. 第 2 正規化

   これは，データベース内で一意な主キーを決定するものである．IFC スキーマの各クラスには必ず GlobalId 属性が存在し，この GlobalId 属性は IFC プロダクトモデルを扱うソフトウェア内で一意な値として定義されているため，これをテーブルの主キーとして利用する．

3. 第 3 正規化

   これは，テーブルに含まれる属性間の推移的関数従属性を排除するものである．第 2 正規化により，テーブルに含まれる属性は主キー属性と非キー属性とに分けられ，主キー属性と非キー属性の間には関数従属性が存在することになる．このとき，異なる非キー属性間に関数従属性が存在すれば，推移的関数従属性が存在すると言う．

   IFC スキーマに推移的関数従属性が存在する場合は，互いに関連し合っている各クラスは互いに同じデータを重複して保持することになる．IFC スキーマはオブジェクト指向分析設計に基いて慎重に設計されているため，このような冗長性は排除されていると考えられる．この仮定が正しければ，クラス毎にテーブルを定義するだけで第 3 正規化を満足させることができる[*1]．

(2) データ型の変換

EXPRESS のデータ型と，標準的なリレーショナル型 DBMS が採用する SQL99 [3] のデータ型には差異があるため，データ型の変換が必要になる．表 4.2 に示すように，SQL99 に対応する

[*1] 表 4.1 で示したように，IFC には多数のクラス定義されており，すべてのクラスで推移的関数従属性がないことを確認することは困難であるため，実験で利用する一部のクラスについてのみ，推移的関数従属性がないことを確認した．

型があればそれを用いるが，EXPRESS の論理型と列挙型には SQL99 に対応する型が存在しないため，文字列で表現し，プログラム側で型を判断することにする．集合体データ型の場合は，すべて別のテーブルとして定義する．定義データ型と選択データ型の場合は，それぞれ，元となるデータ型と選択されたデータ型の種類によって表現方法を決める．どちらも，単純データ型，エンティティデータ型，列挙データ型のいずれかに収斂するため表 4.2 の方法で表現できる．

(3)　関係モデルへの変換手順

以上から，IFC プロダクトモデルを第 3 正規形を満足した関係モデルに変換する手順は次のように整理できる．

1. クラス毎にテーブルを定義する．その際のテーブル名はクラス名と同じにする．
2. クラスの属性をテーブルのカラムとして定義する．その際のカラム名はクラスの属性名と同じにし，GlobalId カラムを主キーとする．
3. 集合体データ型による関連情報を保持するための Relation テーブルを定義し，主キーである GlobalId カラムと関連先の主キーを格納する RelatedObject カラムを定義する．
4. 以下の規則で変換先カラムのデータ型を決める．
   - 主キーである GlobalId カラムは文字列型とする．
   - 変換元属性が単純データ型である場合は，表 4.2 に示した型にする．
   - 変換元属性が集合体データ型である場合は，Relation テーブルへの外部キーとし文字列型にする．
   - 変換元属性がエンティティ型である場合は，関連先関係への外部キーとし文字列型に

表 4.2　関係モデルにおけるデータ型変換方法

EXPRESSのデータ型		関係モデルにおける表現
単純データ型	数値	実数（REAL）
	実数	実数（REAL）
	整数	整数（INTEGER）
	文字列	文字列（NATIONAL CHARACTER VARYING）
	論理型	文字列（CHARACTER VARYING）
	ブール	ブール（BOOLEAN）
	2進	ビット列（BIT）
集合体データ型	配列	関係（TABLE）
	リスト	
	多重重合	
	集合	
名前付きデータ型	エンティティ	関係（TABLE）
	定義	※元となる型の種類に依存
構成データ型	列挙	文字列（CHARACTER VARYING）
	選択	※選択された型の種類に依存

する．

- 変換元属性が列挙型である場合は，文字列型とする．
- 変換元属性が定義型，選択型である場合は，定義内容に従い上記規則のいずれかを適用する．

PC 箱桁橋プロダクトモデル（図 4.2）を変換した結果を図 4.3 に示す．IFC プロダクトモデルの階層構造が，関係モデルではテーブル同士の結合として表現されることが分かる．なお，この 4 つのテーブルに推移的関数従属性は見られない．

4.2.2 半構造データへの変換

XML データベースやドキュメント指向データベースは，そのデータモデルに XML や JSON といった半構造データを採用している．IFC プロダクトモデルをこれらデータフォーマットに変換できれば，IFC プロダクトモデルをデータベースに格納することが可能になる．

(1) XML への変換手法

XML (eXtensible Markup Language) はデータ交換フォーマットの一種であり，W3C (World Wide Web Consortium) により標準化されている [4]．さらに，プロダクトモデルの交換用フォーマットとして ISO10303-28 (STEP-XML) [5] が標準化されており，XML を用いて IFC プロダクトモデルを完全に記述することができる．

IFC プロダクトモデル（図 4.2）の XML による変換例を図 4.4 に示す．

IfcBridge

GlobalId	OwnerHistory	Name	Description	StructureIndicator	CompositionType	StructureType
HXGKy49hQ7+r6h0f3Nt9pQ		PC箱桁橋		BOX_GIRDER_BRIDGE	COMPLEX	COMPOSITE

IfcRelAggregates

GlobalId	OwnerHistory	Name	Description	RelatingObject	RelatedElements
7ET0zbEvSAC8onM3i9kebw				HXGKy49hQ7+r6h0f3Nt9pQ	zhlZjk5OGJlZjJiNzU1Yjc

Relation

GlobalId	IfcBridgePart
zhlZjk5OGJlZjJiNzU1Yjc	MHmkXG1OTNuirgG2T1i08g
zhlZjk5OGJlZjJiNzU1Yjc	nmVdSHIzRWu/6h9L/b1/eA
zhlZjk5OGJlZjJiNzU1Yjc	DQ9PeM7OQ4eoNsV9E+x+SA
zhlZjk5OGJlZjJiNzU1Yjc	kEWTGkwERg2sOZiMJxwABw

IfcBridgePart

GlobalId	Owner History	Name	Description	Structure Indicator	Composition Type	Structure Element Type	TechnoElementType
MHmkXG1OTNuirgG2T1i08g		A1		COMPOSITE	COMPLEX	PIER	MASSIVE_SECTION_ELEMENT
nmVdSHIzRWu/6h9L/b1/eA		P1		COMPOSITE	COMPLEX	PIER	MASSIVE_SECTION_ELEMENT
DQ9PeM7OQ4eoNsV9E+x+SA		A2		COMPOSITE	COMPLEX	PIER	MASSIVE_SECTION_ELEMENT
kEWTGkwERg2sOZiMJxwABw				COMPOSITE	COMPLEX	DECK	UNICELLULAR_MONO_BOX-GIRDER

図 4.3 PC 箱桁橋プロダクトモデルの関係モデルによる表現

```
<?xml version="1.0"?>
<IfcBridge id="1">
  <GlobalId>HXGKy49hQ7+r6h0f3Nt9pQ</GlobalId>
  <Name>PC箱桁橋 </Name>
  <StructureIndicator>BOX_GIRDER_BRIDGE</StructureIndicator>
  <CompositionType>COMPLEX</CompositionType>
  <StructureType>COMPOSITE</StructureType>
</IfcBridge>
<IfcRelAggregates id="2">
  <GlobalId>7ET0zbEvSAC8onM3i9kebw</GlobalId>
  <RelatingObject ref="1" />
  <RelatedElements>
    <IfcBridgePart id="3">
      <GlobalId>MHmkXG1OTNuirgG2T1i08g</GlobalId>
      <Name>A1</Name>
      <StructureIndicator>COMPOSITE</StructureIndicator>
      <CompositionType>COMPLEX</CompositionType>
      <StructureElementType>PIER</StructureElementType>
      <TechnoElementType>MASSIVE_SECTION_ELEMENT</
TechnoElementType>
    </IfcBridgePart>
    <IfcBridgePart id="4">
      <GlobalId>nmVdSHIzRWu/6h9L/b1/eA</GlobalId>
      <Name>P1</Name>
      <StructureIndicator>COMPOSITE</StructureIndicator>
      <CompositionType>COMPLEX</CompositionType>
      <StructureElementType>PIER</StructureElementType>
      <TechnoElementType>MASSIVE_SECTION_ELEMENT</
TechnoElementType>
    </IfcBridgePart>
    <IfcBridgePart id="5">
      <GlobalId>DQ9PeM7OQ4eoNsV9E+x+SA</GlobalId>
      <Name>A2</Name>
      <StructureIndicator>COMPOSITE</StructureIndicator>
      <CompositionType>COMPLEX</CompositionType>
      <StructureElementType>PIER</StructureElementType>
      <TechnoElementType>MASSIVE_SECTION_ELEMENT</
TechnoElementType>
    </IfcBridgePart>
    <IfcBridgePart id="6">
      <GlobalId>kEWTGkwERg2sOZiMJxwABw</GlobalId>
      <StructureIndicator>COMPOSITE</StructureIndicator>
      <CompositionType>COMPLEX</CompositionType>
      <StructureElementType>DECK</StructureElementType>
      <TechnoElementType>UNICELLULAR_MONO_BOX-GIRDER</
TechnoElementType>
    </IfcBridgePart>
  </RelatedElements>
</IfcRelAggregates>
```

図 4.4　IFC プロダクトモデルの XML による変換例

(2) JSON への変換手法

JSON はデータ交換フォーマットの一種であり，IETF (Internet Engineering Task Force) で RFC 4627 として標準化されている [6]．JSON で表現可能なデータ型を表 4.3 に示す．関係モデルへの変換の場合と同様に，EXPRESS のデータ型と JSON のデータ型には差異があるため，データ型の変換が必要になる．表 4.4 に示すように，JSON に対応する型があればそれを用いるが，存在しない場合は文字列で表現して，取得する際にプログラム側で型を判断することにする．集合体データ型の場合は，すべて集合 (array) で表現する．定義データ型と選択データ型の場合は，関係モデルへの変換の場合と同様に，元となるデータ型と選択されたデータ型の種類によって表現方法を決める．JSON 形式はオブジェクト指向に基いているため，オブジェクト同士の関連を OOPL (Object Oriented Programming Language) と同様に表現できる．よって，データ型さえ変換すれば IFC プロダクトモデルを容易に変換できる．

IFC プロダクトモデル（図 4.2）の JSON による変換例を図 4.5 に示す．

表 4.3 JSON データ型

データ型	意味
オブジェクト (object)	順序付けされないstringとvalueのペアの集合
集合 (array)	順序付けされたvalueの集合
値 (value)	string, number, boolean, object, array, true, false, null のいずれか
文字列 (string)	文字列
数値 (number)	数値

表 4.4 JSON 形式へのデータ型変換方法

EXPRESSのデータ型		JSONにおける表現
単純データ型	数値	数値 (number)
	実数	数値 (number)
	整数	数値 (number)
	文字列	文字列 (string)
	論理型	文字列 (string)
	ブール	true, またはfalse
	2進	文字列 (string)
集合体データ型	配列	集合 (array)
	リスト	
	多重重合	
	集合	
名前付きデータ型	エンティティ	オブジェクト (object)
	定義	※元となる型の種類に依存
構成データ型	列挙	文字列 (string)
	選択	※選択された型の種類に依存

```
{
  "GlobalId" : "7ET0zbEvSAC8onM3i9kebw",
  "RelatingStructure": {
    "StructureIndicator" : "COMPOSITE",
    "CompositionType" : "COMPLEX",
    "GlobalId" : "HXGKy49hQ7+r6h0f3Nt9pQ",
    "Name" : "PC箱桁橋",
    "StructureType" : "BOX_GIRDER_BRIDGE"
  },
  "RelatedElements":[
    {
      "StructureIndicator" : "COMPOSITE",
      "CompositionType" : "COMPLEX",
      "GlobalId" : "MHmkXG10TNurgG2T1i08g",
      "Name" : "A1",
      "StructureElementType" : "PIER",
      "TechnoElementType" : "MASSIVE_SECTION_ELEMENT"
    },
    {
      "StructureIndicator" : "COMPOSITE",
      "CompositionType" : "COMPLEX",
      "GlobalId" : "nmVdSHIzRWu/6h9L/b1eA",
      "Name" : "P1",
      "StructureElementType" : "PIER",
      "TechnoElementType" : "MASSIVE_SECTION_ELEMENT"
    },
    {
      "StructureIndicator" : "COMPOSITE",
      "CompositionType" : "COMPLEX",
      "GlobalId" : "DQ0PeM7OQ4eoNsV9$+x+SA",
      "Name" : "A2",
      "StructureElementType" : "PIER",
      "TechnoElementType" : "MASSIVE_SECTION_ELEMENT"
    },
    {
      "StructureIndicator" : "COMPOSITE",
      "CompositionType" : "COMPLEX",
      "GlobalId" : "kEWTGkwERg2sOZiMJxwABw",
      "StructureElementType" : "DECK",
      "TechnoElementType" : "UNICELLULAR_MONO_BOX-GIRDER"
    }
  ]
}
```

図 4.5　IFC プロダクトモデルの JSON による変換例

4.2.3 プロパティグラフモデルへの変換

IFC のクラスが実体化したインスタンスをノードに，インスタンス間の関連をエッジに対応させると，グラフに近い構造になるため，ほぼ元の構造そのままに変換可能である．しかし，実験で利用するグラフ型 DBMS 製品である Neo4j [7] で利用可能な型と EXPRESS の型には相違があるため，JSON の場合と同様に型の表現方法を定める必要がある．プロパティグラフモデルには標準が定められていないため，型などの仕様は製品に依存するのが実態である．表 4.5 に Neo4j で利用可能な型の種類を示す．また，EXPRESS 固有の型を Neo4j で利用可能な型に変換する方法を表 4.6 のように定めた．

表 4.6 の変換方法と以下のルールで IFC プロダクトモデルをプロパティグラフモデルに変換する．

表 4.5 Neo4j DBMS のデータ型

データ型	意味
整数	整数
実数	実数
文字列	文字列
ブール	真 (true)，または偽 (false)
配列	整数，実数，文字列，ブールの順序付けられた集合．同一型の要素のみ含む．

表 4.6 プロパティグラフモデルへのデータ型変換方法

EXPRESSのデータ型		プロパティグラフモデルにおける表現
単純データ型	数値	実数
	実数	実数
	整数	整数
	文字列	文字列
	論理型	文字列
	ブール	ブール
	2進	文字列
集合体データ型	配列	配列，またはノード
	リスト	
	多重重合	
	集合	
名前付きデータ型	エンティティ	ノード
	定義	※元となる型の種類に依存
構成データ型	列挙	文字列 (string)
	選択	※選択された型の種類に依存

1. 各クラスに対応したノードを作成する．その際にノードのラベルにエンティティ名を格納する．
2. クラスの属性が単純データ型，列挙データ型である場合は，ノードに属性名と同名のプロパティを作成して属性値を格納する．
3. クラスの属性が集合体データ型であり，集合要素のデータ型が単純データ型，列挙データ型である場合は，ノードに属性名と同名のプロパティを作成して集合要素の配列を格納する．
4. クラスの属性が集合体データ型であり，集合要素のデータ型がクラス型である場合は，各集合要素に対応するノードを作成して，関連元クラスから関連先クラスへの方向性を持つエッジで接続する．集合体データ型に順序がある場合は，エッジに index プロパティを作成してインデックス値を格納する．その後，name プロパティを作成して属性名を格納する．

以上の手順で，図 4.2 の IFC プロダクトモデルを変換した結果を図 4.6 に示す．プロダクトモデルの構造をほぼそのままに表現できていることが分かる．

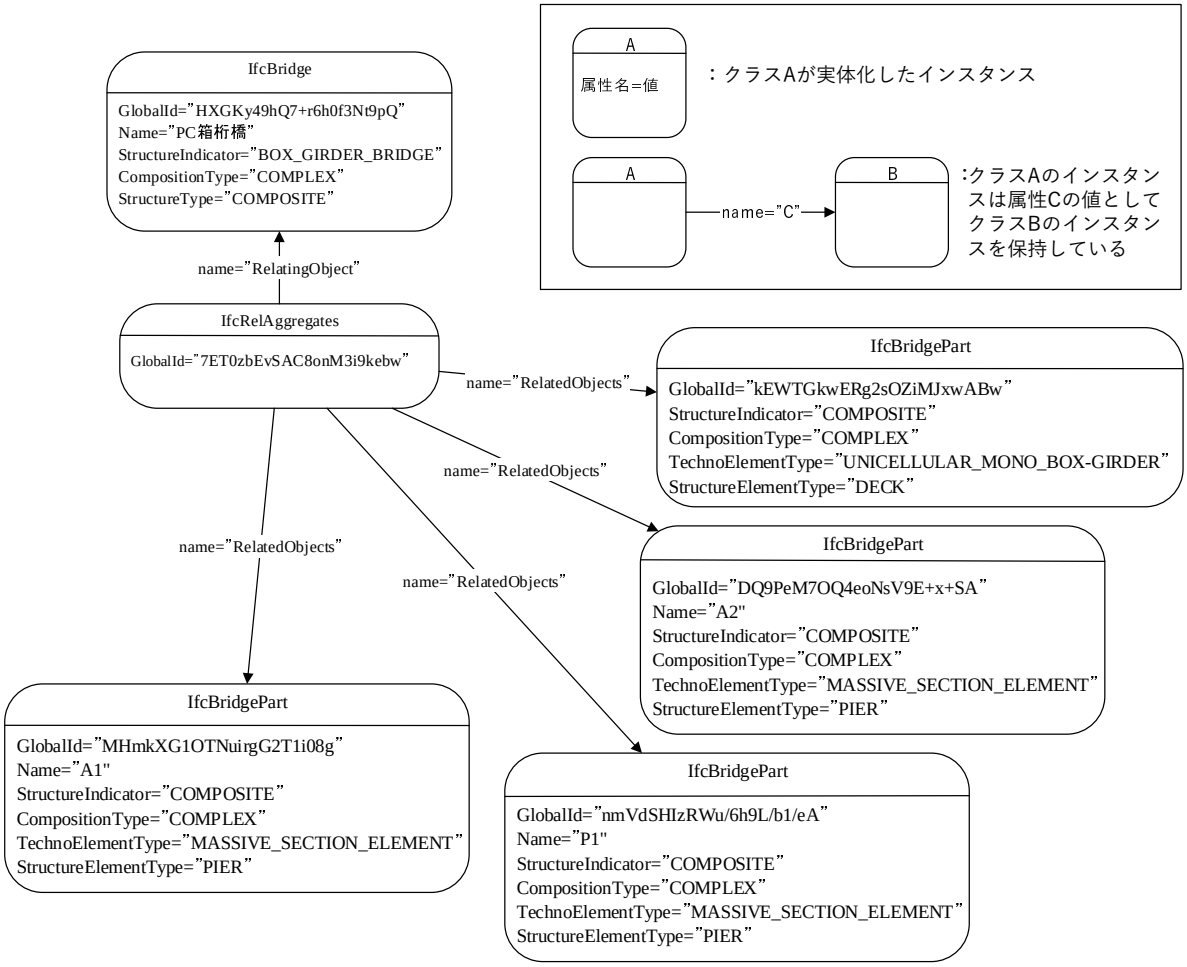

図 4.6　PC 箱桁橋プロダクトモデルのプロパティグラフモデルによる表現

4.2.4 キー・バリューモデルへの変換

第 3 章で述べたように，キー・バリューモデルは非常に単純な構造であり，1 つのキーに対して 1 つの値を格納することしかできない．そのため，複雑な階層構造を有する IFC プロダクトモデルを格納するのは困難であるが，どうにかして格納するとすれば，2 つのアプローチが考えられる．1 つは，何かしらの表記法を用いて IFC プロダクトモデルをテキストで記述し，記述したテキストを 1 つのバリューとして格納する方法である．もう 1 つは，キー・バリューモデルで表現できるまで，IFC プロダクトモデルを細分化して格納する方法である．

以下で，これらの手法について詳しく検討する．

(1) プロダクトモデルのテキスト表現をバリューとする手法

この手法は，IFC プロダクトモデルをテキストで表現できれば実現可能である．IFC プロダクトモデルは Part21 形式で完全に記述することが可能であるし，また，データ型の表現に制限はあるものの，一般的にデータ交換フォーマットとして利用されている XML や JSON といったデータフォーマットで記述することも可能である．XML，または JSON を用いるとすれば，変換手法は 4.2.2 項で示した変換手法をそのまま利用できる．

ただし，いずれのフォーマットで記述したとしても，キー・バリュー型 DBMS はこれらフォーマットを理解しないため，データ取得時はプログラムでモデル構築処理を実装する必要がある．また，検索処理などの，一般的には DBMS の機能を用いて実現可能な処理を，プログラムで実装する必要が生じる．実現は容易であるがデメリットが多い手法である．キー・バリュー型 DBMS を用いなくてはならない制約がないのであれば，同様のアプローチで DBMS の機能を利用可能なドキュメント指向型 DBMS を用いるべきである．

(2) キー・バリューモデルを用いて新たなデータモデルを構築する手法

キー・バリュー型の派生型の 1 つであるカラム指向型のデータモデルに変換することを考える．カラム指向型では，1 レコードにカラムとバリューのペアを複数格納できるため，クラスの属性名のカラムに属性値を格納すれば 1 レコードに 1 インスタンスの情報を格納することができる．

この手法を用いれば，IFC プロダクトモデルを構成する多数のインスタンスをキー・バリュー型 DBMS に格納することが可能だが，キー・バリュー型 DBMS はレコード同士を関連付ける機能を持っていないため，IFC プロダクトモデルを再構成することができない．そのため，図 4.7 のように各レコードに関連先のキーを格納しておき，プログラムによりレコード同士の関連を判断して再構成することになる．

この手法で得られる，複数の属性を持ったレコードが関連先レコードに方向性を持って直接接続するデータ構造はプロパティグラフモデルと類似している．よって，カラム指向型 DBMS に格納する場合の変換手法は，前項で検討したプロパティグラフモデルへの変換手法をそのまま用いれば良い．

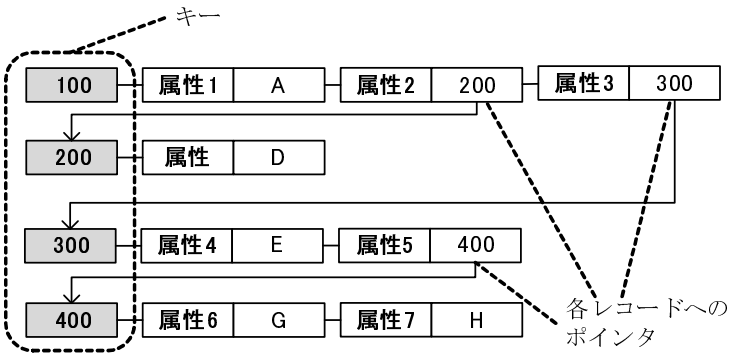

図 4.7　カラム指向型でのインスタンス間の関連の表現

4.3　実験

IFC プロダクトモデルを，4.2 節で考案した変換手法を用いて変換し，以下の DBMS 製品に格納して実験を行なった．

- SQL Server 2008R2 [8]（リレーショナル型）
- MongoDB 3.0.6 [9]（ドキュメント指向）
- Neo4j 2.2.6（グラフ型）

上記製品の選定理由であるが，本論では，BIM/CIM 情報共有システムとしてシステム化することを想定しているため，すでに十分な運用実績があり，長期の連続運用に耐え得る製品から選択するべきだと考えた．しかしながら，NoSQL 製品の実績はまだ少なく判断が困難であったため，MongoDB と Neo4j の選択に関しては，各製品の現時点における利用状況[*2]を元にして選択した．十分な運用実績があることの直接の証拠にはならないが，広く使われている DBMS における実験結果の方が価値が高いと言えるし，また，今後 NoSQL DBMS の仕様が標準化されるとすれば，広く使われている DBMS の仕様が標準に取り込まれる可能性が高いため，利用状況による選択も意味はあると考える．いずれにしても，本章ではデータモデルの違いに着目しており，DBMS 製品の違いは検討対象外であるが，仮に他の DBMS を用いた場合の結果については，4.4 節で考察する．

表 4.7 に，実験の実行環境を示す．スペックが異なる複数のサーバを利用しているが，本実験では各 DBMS 毎の実行時間の変化に着目しており，実行時間そのものの比較は行なわないため，実験結果には影響しない．また，計測は各条件毎に 5 回以上実施してその平均値を計測結果としている．その際，DBMS が持つ特殊な高速化機能が働いて，測定結果のばらつきが大きくなる場合

[*2] DB-ENGINES ウェブサイトの DB-Engines Ranking を元にした．MongoDB と Neo4j は，同カテゴリ内では圧倒的にスコアが高い（`https://db-engines.com/en/ranking/document+store`, `https://db-engines.com/en/ranking/graph+dbms`）．本サイトでは，様々な要素から各 DBMS 製品のスコアを総合的に算出している [10].

表 4.7 実行環境

DBサーバ	項目	スペック等
・Neo4j ・MongoDB	CPU	Intel Core i5-2520M（2.5GHz, 2コア）
	メモリ	16GB（DDR3 133MHz）
	ストレージ	500GB HDD
	OS	Mac OS X 10.10 64bit
・SQL Server	CPU	Intel Xeon （3.60GHz, 4コア）
	メモリ	4GB（DDR 166MHz）
	ストレージ	160GB HDD
	OS	Microsoft Windows Server 2003 Standard Edition SP2 32bit

があり，その場合は該当機能を抑止する等の対処をした[*3]．なお，実験で使用したリレーショナル DBMS のスキーマ（テーブル定義）は付録 B に示した．

キー・バリュー型については，前述したようにドキュメント指向型，もしくはグラフ型と同等のデータモデルに変換することになるため実験は行なわない．

4.3.1 実験対象データ

BIM/CIM で取り扱うプロダクトモデルはデータ量が膨大で，かつ階層構造が深いという特徴があるため，格納データ量と取得データの階層の深さに着目して実験を行なった．対象データの一部分を UML オブジェクト図で記述したものを図 4.8 に示す．これより，対象データが階層の深い構造になっていることが分かる．

4.3.2 実験 1：データ数に着目した実験

(1) 実験内容

各 DBMS に格納するインスタンスの数を段階的に増やしていき，そこから 1 本の鉄筋の位置情報（図 4.8 のオレンジ色のインスタンス）を取得する際の実行時間の変化を計測する．

(2) 対象データ

PC 箱桁橋の一部をプログラムで自動生成したものをテストデータとして用いた．PC 橋梁の場合は，桁が施工単位毎にモデリングされるため，橋長が長くなるに従い桁を構成する部材の数（データ量）が増える．また，IFC プロダクトモデルでは，形状の位置データを格納する IfcCartesianPoint クラスのインスタンス数がもっとも多くなる傾向にある．そこで，1 施工単位に約 26,000 個のインスタンス（多くを鉄筋形状位置データが占める）を含むデータを生成し，施工単位を増やすことでデータを増加させた．

[*3] SQL Server の場合は，同じクエリを連続して実行すると，実行結果がキャッシュされ，2 回目以降の処理速度が極端に速くなる場合があったため，キャッシュを無効にするオプションを設定して実施した．

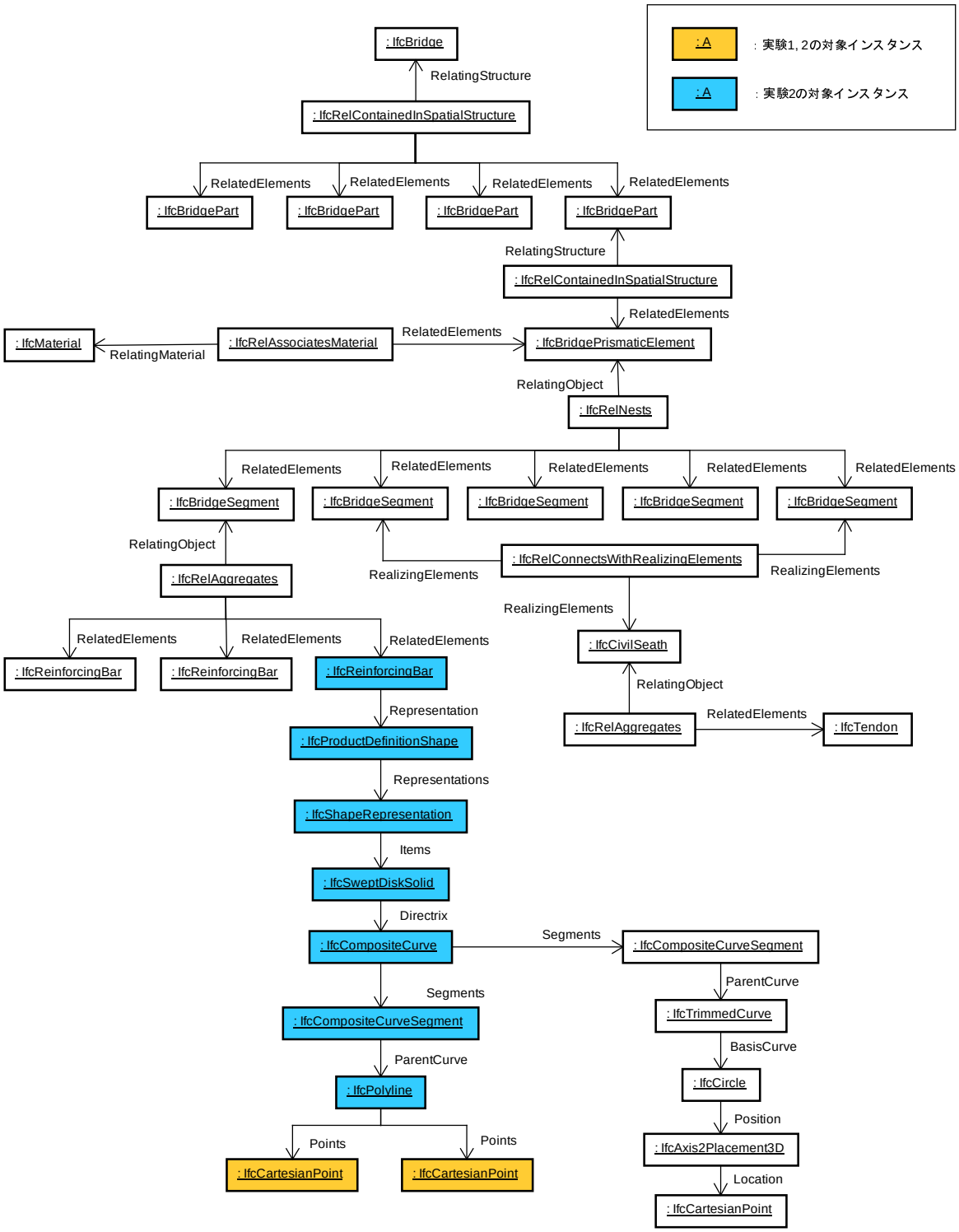

図 4.8　実験対象データ（UML オブジェクト図）

(3) 実行クエリ

SQL Server, Neo4j のそれぞれで実行したクエリを，図 4.9，図 4.10 に示す．また， Neo4j のクエリ言語である Cypher 言語のデータベースの参照に関する文法を表 4.8 に示す．データベースへの追加等も含めた詳細な言語仕様は参考文献 [11] を参照されたい．

(4) 実験結果

結果を図 4.11 に示す．SQL Server はデータ量の増加に伴なって処理時間が増えるが，Neo4j は格納するデータ数によらず，一定時間でデータを取得できることが分かる．MongoDB の場合は，1 ドキュメントのサイズが 16M バイトに制限されており [9] ，クラス数を 12,000 個までしか増やすことができなかったため，処理時間の変化を計測することができなかった．

4.3.3 実験 2：階層数に着目した実験

(1) 実験内容

各 DBMS に IFC プロダクトモデルを格納し，格納データ数を固定した状態で，1 本の鉄筋情報の取得にかかる処理時間を計測する．取得対象のインスタンスは図 4.8 中の青色とオレンジ色のインスタンスであるが，取得する鉄筋情報の階層を，IfcReinforcingBar, IfcProductionDefinitionShape, ..., IfcCartesianPoint インスタンスまで増やしながら計測する．

(2) 対象データ

実験 1 と同様に PC 箱桁橋のプロダクトモデル利用した．各階層毎の取得対象クラスとその格納全インスタンス数，および取得対象インスタンス数を表 4.9 に示す．階層が深くなるにつれインスタンス数が増えることが分かる．ただし，MongoDB では 1 ドキュメント 16M バイトの制限があ

```
1 SELECT * FROM IfcCartesianPoint CP
2   INNER JOIN Relation R ON R.GlobalId=CP.Coordinates
3   INNER JOIN IfcLengthMeasure LM on LM.GlobalId=R.RelatedObject
4  WHERE CP.GlobalId in ('71d630ab6be1406ba4e962ba31422935','
  fdee0a5a82774b02b67170b799e91569')
```

図 4.9 実験 1 のクエリ（SQL Server 用）

```
1 START a=NODE(45)
2 MATCH (a)-[:SET]->(b)-[*]->(c)
3  WHERE c.GlobalId="71d630ab6be1406ba4e962ba31422935"
4   OR c.GlobalId="fdee0a5a82774b02b67170b799e91569"
5  RETURN a,b,c;
```

図 4.10 実験 1 のクエリ（Neo4j 用）

表 4.8　Neo4j のクエリ言語である Cypher の文法（参照に関する仕様のみ）

参照クエリの文法（※[ ]内は省略可能）	
[MATCH パターン WHERE 条件] [WITH 識別子] RETURN 識別子	
キーワード	
キーワード	意味
MATCH パターン	パターンに合致する部分グラフを指定する.
WHERE 条件	条件に合致するパターンに限定する.
WITH 識別子	識別子で指定された部分グラフ内のノード，またはエッジをクエリの次のパートに渡す.
RETURN 識別子	識別子で指定された部分グラフ内のノード，またはエッジを返す.
ORDER BY 識別子	識別子で指定されたノード，またはエッジで結果をソートする.
パターン	
パターン	意味
(n:Person)	Personラベルを持つノード.
(n {name: "Smith"})	nameプロパティが"Smith"であるノード.
(n)-->(m)	ノードnからノードmへの接続.
(n)--(m)	ノードnとノードmの方向性のない接続.
(n)-[:KNOWS]->(m)	ノードnからノードmへの"KNOWS"関係による接続.
(n)-[r]->(m)	ノードnからノードmにエッジrで接続している.
(n)-[*1..5]->(m)	1〜5までの長さのパスで接続されたノードnからノードmのグラフ.
(n)-[*]->(m)	任意の長さのパスで接続されたノードnからノードmのグラフ.

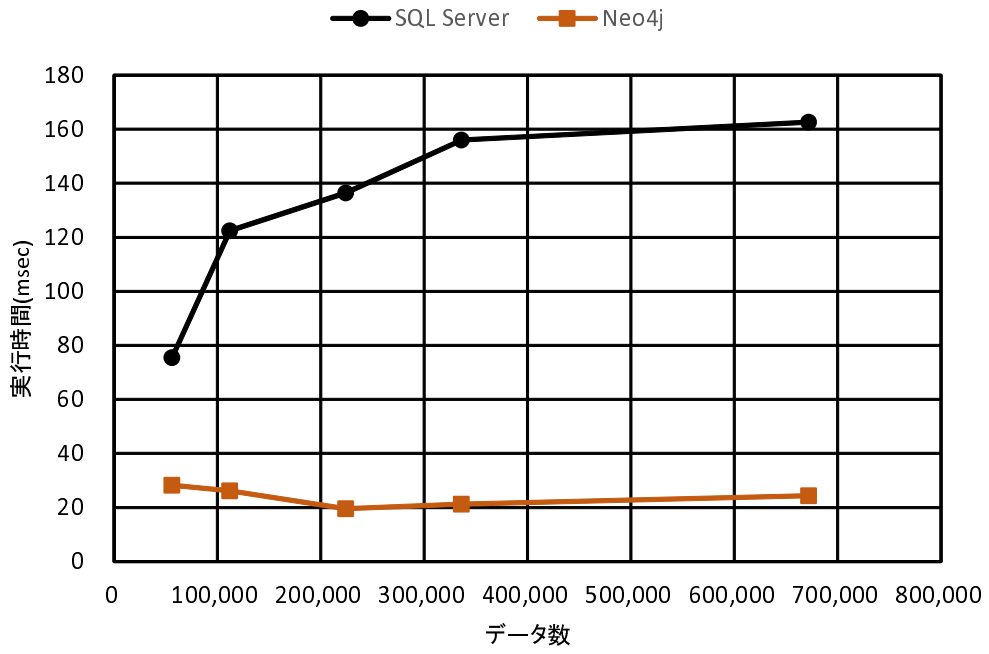

図 4.11　実験 1 の測定結果

るため，格納データ数を 1/10 にして実験した．

(3) 実行クエリ

SQL Server, Neo4j, MongoDB のそれぞれで実行したクエリを，図 4.12–4.14 に示す．また，MongoDB のクエリの参照に関する文法を表 4.10 に示す．データベースへの追加等も含めた詳細な言語仕様は参考文献 [9] を参照されたい．

(4) 実験結果

結果を図 4.15 に示す．SQL Server は階層が増えるに従い処理時間が大きく増加しているが，Neo4j, MongoDB では取得データの階層数によらず，ほぼ一定時間でデータを取得できている．

表 4.9 実験 2 の階層別対象データ

階層	エンティティ	格納数	取得数
1	IfcReinforcingBar	18,000	1
2	IfcProductDefinitionShape	18,000	1
3	IfcShapeRepresentation	18,000	1
4	IfcSweptDiskSolid	18,000	1
5	IfcCompositeCurve	18,000	1
6	IfcCompositeCurveSegment	144,000	8
7	IfcPolyline	144,000	8
8	IfcCartesianPoint	288,000	16

```
1  SELECT * FROM IfcReinforcingBar RB
2    INNER JOIN IfcProductDefinitionShape PDS ON PDS.GlobalId=RB.
   Representation
3    INNER JOIN Related R5 ON R5.GlobalId=PDS.Representations
4    INNER JOIN IfcShapeRepresentation SR ON SR.GlobalId=R5.
   RelatedObject
5    INNER JOIN Related R6 ON R6.GlobalId=SR.Items
6    INNER JOIN IfcSweptDiskSolid SDS ON SDS.GlobalId=R6.
   RelatedObject
7    INNER JOIN IfcCompositeCurve CC ON CC.GlobalId=SDS.Directrix
8    INNER JOIN Related R7 ON R7.GlobalId=CC.Segments
9    INNER JOIN IfcCompositeCurveSegment CCS ON CCS.GlobalId=R7.
   RelatedObject
10   INNER JOIN IfcPolyline PL ON PL.GlobalId=CCS.ParentCurve
11   INNER JOIN Related R8 ON R8.GlobalId=PL.Points
12   INNER JOIN IfcCartesianPoint CP ON CP.GlobalId=R8.RelatedObject
13  WHERE RB.GlobalId='765e9e1859d04c788b7f2a9d0e7af079'
```

図 4.12 実験 2 のクエリ（SQL Server 用）

```
START a=NODE(38) MATCH (a)-->(b) RETURN a,b;
START a=NODE(38) MATCH (a)-[*..2]->(b) RETURN a,b;
START a=NODE(38) MATCH (a)-[*..3]->(b) RETURN a,b;
START a=NODE(38) MATCH (a)-[*..4]->(b) RETURN a,b;
START a=NODE(38) MATCH (a)-[*..5]->(b) RETURN a,b;
START a=NODE(38) MATCH (a)-[*..6]->(b) RETURN a,b;
START a=NODE(38) MATCH (a)-[*..7]->(b) RETURN a,b;
START a=NODE(38) MATCH (a)-[*..8]->(b) RETURN a,b;
```

図 4.13　実験 2 のクエリ（Neo4j 用）

```
db.getCollection('size10num100000').find({},{"bridge000001":{
$slice:[37,1]},
  "bridge000001.relatingStructure.IfcReinforcingBar":1})
db.getCollection('size10num100000').find({},{"bridge000001":{
$slice:[37,1]},
  "bridge000001.relatingStructure.IfcProductDefinitionShape":1})
db.getCollection('size10num100000').find({},{"bridge000001":{
$slice:[37,1]},
  "bridge000001.relatingStructure.IfcShapeRepresentation":1})
db.getCollection('size10num100000').find({},{"bridge000001":{
$slice:[37,1]},
  "bridge000001.relatingStructure.IfcSweptDiskSolid":1})
db.getCollection('size10num100000').find({},{"bridge000001":{
$slice:[37,1]},
  "bridge000001.relatingStructure.IfcCompositeCurve":1})
db.getCollection('size10num100000').find({},{"bridge000001":{
$slice:[37,1]},
  "bridge000001.relatingStructure.IfcCompositeCurveSegment":1})
db.getCollection('size10num100000').find({},{"bridge000001":{
$slice:[37,1]},
  "bridge000001.relatingStructure.IfcPolyline":1})
db.getCollection('size10num100000').find({},{"bridge000001":{
$slice:[37,1]},
  "bridge000001.relatingStructure.IfcCartesianPoint":1})
```

図 4.14　実験 2 のクエリ（MongoDB 用）

表 4.10　MongoDB のクエリ言語の文法（参照に関する仕様のみ）

参照クエリの文法（条件に合致したドキュメントを返す）	
db.collections.find(条件, ...)	
条件	
パターン	意味
{フィールド: 値}	「フィールド（属性）」の内容と「値」が同値かどうか比較する.
{フィールド:{演算子: 値}}	「フィールド（属性）」の内容と「値」を演算子で比較する.

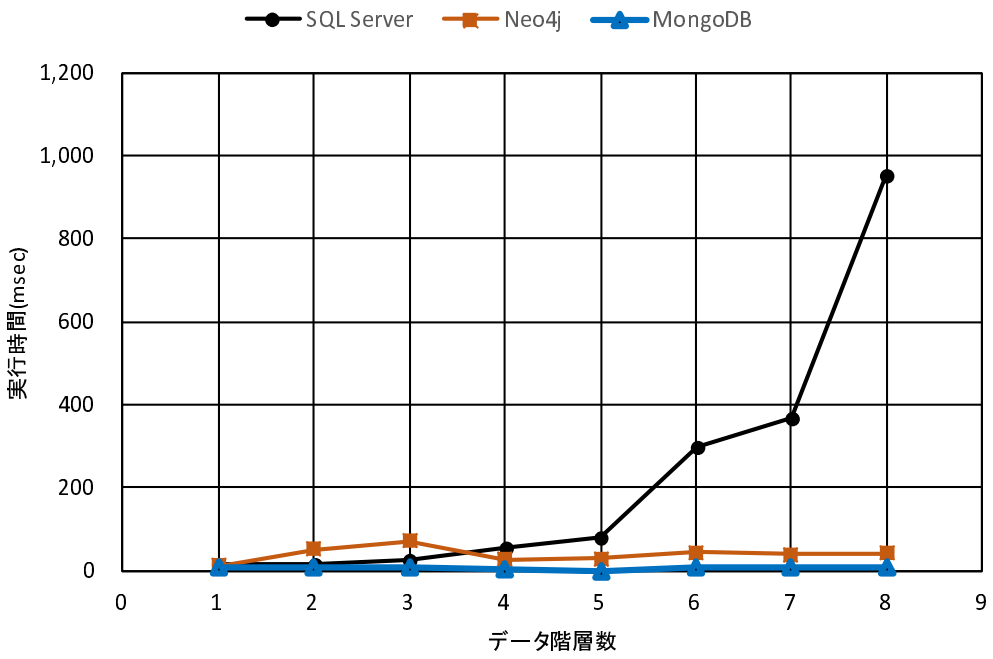

図 4.15　実験 2 の測定結果

4.4　考察

前節で，格納データ数と取得対象の階層が増加した場合の，各 DBMS でのデータ取得処理時間を計測した．それらの結果に対しての考察を以下に述べる．

4.4.1　リレーショナル型

リレーショナル型は，二分木と呼ばれるデータ構造で格納データを管理しており，n 個のデータを含む関係からデータを取得（検索）する計算量は以下になる．

$$O(\log_2 n) \tag{4.1}$$

また，階層構造があるデータを取得するには，階層分の検索処理が必要になるため，各階層に相当する関係に含まれるデータ数をそれぞれ，$n1, n2, ...$ とすれば以下になる．

$$O(\log_2 n1 + \log_2 n2 + ...) \tag{4.2}$$

実験 1 では，階層数を 1 に固定してデータ数を増加させたため，対象データ数を n とすると，$O(\log_2 n)$ で処理時間が増加している．一方，実験 2 では $n1, n2, ..., n8$ が固定であり，かつ $n2, ..., n8$ はそれぞれ $n1$ に比例しているため，階層数を m とすれば $O(m)$ で一様に処理時間が増加するとみなすことができる．しかし，IFC プロダクトモデルでは階層数 m は高々数十程度にし

かならないため，実際には一様には増えず，各階層に対応するインスタンス数の違いに依存して処理時間が増加する．表 4.9 と図 4.15 を見ると，格納インスタンス数が多い階層 6〜8 で処理時間が大きく増加していることがわかる．

式 4.1 は，リレーショナル DBMS の内部スキーマである二分木データ構造の特徴であり，本章における実験結果は，理論通りの結果が出ていると言える．そのため，SQL Server 以外のリレーショナル DBMS を用いて実験を行なったとしても，同様の結果が出るはずである．

4.4.2　ドキュメント指向

前章で述べたように，MongoDB には 1 ドキュメント 16M バイトという制限が存在するため，実験 1 では意味のある結果を出すことができなかった．16M バイトは約 12,000 インスタンスに相当し，IFC プロダクトモデルは小規模なモデルでも数万インスタンス，大規模なモデルでは数十万から数百万インスタンスに達するため，これは実用に耐えない制限である．

対応方法としては，1 橋梁 1 ドキュメントとせずに，1 インスタンス 1 ドキュメントとして格納する方法が考えられる．その場合のデータ構造は，4.2.4 (2) で示したカラム指向型による表現に近い構造になる（図 4.7）．そこで論じたように，最終的にプロパティグラフモデルに近いデータ構造に収斂するとすれば，グラフ構造の扱いに最適化されているグラフ型 DBMS が有利と考えられる．

階層構造が増えた場合の処理時間は，図 4.15 の通り階層構造に依らず一定時間で処理可能である．実際は取得対象オブジェクト数が増えるため，処理時間も増えるはずだが，MongoDB では対象となるドキュメント全体をメモリに読み込んでから処理するため，1 ドキュメント内の検索処理は非常に高速に動作する [12] ．よって，計測時間に現われなかったと考えられる．この処理時間の速さは MongoDB の優位点の 1 つであるが，ドキュメントサイズの制限とトレードオフの関係がある．

ドキュメント指向 DBMS のデータモデルは，入力可能なデータ形式を定めているだけに過ぎず，内部のデータ構造（内部スキーマ）は各 DBMS 製品に依存しているのが実態である．そのため，ここで述べたデータ取得性能に関する考察は，他のドキュメント指向 DBMS には当て嵌まらないと考えられる．

4.4.3　グラフ型

グラフ型において，e 個の外向きのエッジを持つノードから他のノードを走査する処理時間は，$O(e)$ である．同様にして複数個のノードを走査する処理は，それぞれのノードから出るエッジ数を $e1, e2, ...$ とすれば，$O(e1 + e2 + ...)$ であるが，ノードのエッジ数はスキーマの構造に依存するだけであり，格納データ数 n には依存しない．一般的に，グラフ型 DBMS は数百から数千オーダーのエッジ数であれば非常に高速に動作するよう実装されている．それ以上の数万オーダーのエッジが接続するようなグラフであれば，速度低下が懸念されるが，そのような構造のプロダクトモデルはまれであると考えられる．そのため，2 ノード間の走査に要する処理時間は定数時間とみ

なすことができ，ノードを m 回走査する処理時間は $O(m)$ となる [11].

実験 1 は，全格納データ n が増加しても，走査データ数 m は固定であるため，処理時間も固定となっている．実験 2 では対象データの階層が増加するに伴ない走査回数も増加するため，実行時間も増えるはずだが，計測結果には現れていない．Neo4j は 1 秒間に数百万ノードを処理できる性能を持っているが，実験で使用したデータは 8 階層で 37 ノードしかないため，計測値には現れなかったと考えられる．

グラフデータベースも，ドキュメント指向 DBMS と同様に標準化がなされておらず，発展途上であるため，同じグラフデータベースに分類されていても，Neo4j とは異なる内部データ構造を持つグラフ DBMS が存在している．Neo4j はインデックスなし隣接性と呼ばれる特性を持っており [11]，これは隣接したノードへの参照が定数時間である特性であるが[*4]，このインデックスなし隣接性を持つグラフ DBMS であれば，Neo4j 以外の DBMS であっても同様の実験結果が得られるはずである．

4.5 本章のまとめ

本章の成果，および結論を以下に示す．

4.5.1 データモデル変換手法に関するまとめ

IFC プロダクトモデルを，以下データモデルに変換する手法を検討し，関係モデル，JSON 形式，プロパティグラフモデルについては，変換手法を考案した．

- 関係モデル
- XML 形式
- JSON 形式
- プロパティグラフモデル
- キー・バリューモデル

プロパティグラフモデルと JSON については，変換手法を考案しただけでなく，Java 言語を用いて変換プログラムを作成し，複数の IFC プロダクトモデルの変換を実行している．その際，変換元のインスタンス数を数えることで，変換結果のおおよその正しさを確認したが，プログラムの網羅的なテストは行なっていない．

[*4] この特性を持つグラフデータベースでは，グラフ探索の計算量が，グラフ理論における探索アルゴリズムの時間計算量と同じになるため，グラフ理論と適合性が高いと言える．Neo4j 以外では，IBM System G [13] DBMS が該当する．

4.5.2　DBMS への格納と取得性能に関するまとめ

さらに，これらデータモデル変換手法を用いて，IFC プロダクトモデルを以下の具体的な DBMS 製品に格納することに成功した．

- SQL Server 2008R2
- MongoDB
- Neo4j

その結果，IFC プロダクトモデルをリレーショナル型 DBMS に格納した場合には，対象データの階層構造が深くなるに従いデータ取得処理性能が劣化することを明らかにした．一方，グラフ型 DBMS ではそのような階層構造に依存した性能劣化が発生しないため，グラフ型 DBMS が IFC プロダクトモデルのような階層の深いデータ構造に適している DBMS の 1 つであることも同時に明らかになった．

参考文献

[1] E. Lebegue, B. Fies, J. Gual, G. Arthaud, T. Liebich, N. Yabuki: IFC-BRIDGE V3 Data Model - IFC4 Edition R3, 2013.

[2] 矢吹信喜，李占涛: 日仏橋梁プロダクトモデルの統合化による新 IFC-BRIDGE の開発と CAD コンバータの改良, 土木情報利用技術論文集, Vol.15, pp.59–66, 2006.

[3] ISO/IEC 9075:1999: Database Language SQL, 1999.

[4] W3C: Extensible Markup Language (XML) 1.1 (Second Edition), 2006.

[5] ISO 10303-28:2007: Industrial automation systems and integration – Product data representation and exchange – Part 28: Implementation methods: XML representations of EXPRESS schemas and data, using XML schemas, 2007.

[6] D. Crockford: The application/json Media Type for JavaScript Object Notation (JSON), 2006, `http://www.ietf.org/rfc/rfc4627.txt` (参照 2017/6/1).

[7] Neo Technology, Inc.: Neo4j: The World's Leading Graph Database, `https://neo4j.com/product/` (参照 2017/6/1).

[8] Microsoft Corporation: SQL Server 2008 R2 editions overview, `https://assets.microsoft.com/en-us/SQLServer2008R2EditionsDatasheet\_1.pdf` (参照 2017/6/1).

[9] MongoDB, Inc.: What is MongoDB?, `https://www.mongodb.com/what-is-mongodb` (参照 2017/6/1).

[10] solid IT gmbh: Method of calculating the scores of the DB-Engines Ranking, `https://db-engines.com/en/ranking_definition` (参照 2017/6/26).

[11] Ian Robinson, Jim Webber, Emil Eifrem, 佐藤直生（監訳）木下哲也（訳）: グラフデータベース Neo4j によるグラフデータモデルとグラフデータベース入門, オライリージャパン, 2015.

[12] MongoDB, Inc.: The MongoDB 3.0 Manual, `http://docs.mongodb.org/manual/` (参照 2015/10).

[13] IBM System G Team: About IBM System G, `http://systemg.research.ibm.com/about.html` (参照 2017/6/27).

第 5 章

グラフデータベースによるプロダクトモデル部分抽出手法の開発

5.1　本章の概要

第 1 章で述べたように，プロダクトモデルには建築物や土木構造物のライフサイクル全体で発生する膨大な量のデータが含まれるため，特定の作業フェーズに着目すれば不必要なデータを多く含む冗長なモデルと言える．また，同じ作業フェーズであっても，形状を確認するのか，部材の材料を確認するのか，設備の数量を確認するのか，その都度，要求される情報が異なる．しかし，BIM/CIM ではすべてのデータが 3 次元プロダクトモデルに集約されるため，いずれの作業フェーズにおいても巨大なプロダクトモデルが必要となり，ネットワーク上で共有しようとすると，その都度膨大な量のデータ転送が発生し，効率が悪い．そのため，ネットワーク上におけるプロダクトモデルの共有を実現するには，巨大なプロダクトモデルから，必要な情報のみを抽出する技術が重要になる．

第 4 章で，様々な方式の DBMS に IFC プロダクトモデルを格納する手法を考案し，さらに，他方式の DBMS よりもグラフ型 DBMS にメリットが多いことを示した．よって，本章ではグラフ型 DBMS にプロダクトモデルを格納し，そこから必要な情報のみを抽出する手法を開発する．

5.2　部分抽出手法の開発

グラフ型 DBMS に格納した IFC プロダクトモデルから，モデルの一部分のみを抽出する手法を検討する．本章では，BIM/CIM におけるプロダクトモデルの利活用で最も需要が高いと思われる形状データの抽出手法と，特定フロアや部材等の部分データを抽出する手法を検討することにした．IFC プロダクトモデルからプロパティグラフモデルへの変換手法は，基本的には第 4 章で考案した手法を用い，必要があれば変更を加える．

これらの 2 種類の部分抽出処理は，図 5.1 に示したサンプルモデルに当てはめると，前者がグラフ全体に散らばって存在する（形状）情報を収集する処理であるのに対して，後者がグラフ全体か

ら局所的に存在する情報（例えば図 5.1 の右端に位置する材料情報等）を特定する処理になる．どのような種類の情報を抽出するにしても，グラフ型データベースにおけるデータ抽出処理は，これら 2 種類の抽出処理の組み合わせになると考えられる．

5.2.1　部分抽出アルゴリズム

(1)　分散したデータの抽出（全形状データの抽出）

IFC プロダクトモデル全体に分散して格納されているデータを抽出することを考える．説明のため，モデル全体の形状データを抽出する場合を例にする．

IFC スキーマでは，物理的な形状を持つエンティティは IfcProduct クラスを継承するように定義されているため（図 5.2），IfcProduct の派生クラスのインスタンスとそれに関連するインスタンス（図 5.1 の網掛けのノード）を抽出すれば良い．これは，IfcProduct の派生クラスのインスタンスを開始ノードとし，IfcCartesianPoint インスタンスを末端のノードとする部分グラフの集合となる．

これを整理すると以下になる．

1. 探索開始ノードをプロジェクト名により決定する．
2. 1 で決定したノードに接続されている全ノードのうち，IfcProduct 派生クラスのノードを抽出する．
3. 2 に含まれる形状データ（形状種類，および位置情報）を抽出する．

1 に関して，IFC ではモデリング対象をプロジェクトとして管理することになっており，1 モデルにつき必ず，プロジェクトを表わす IfcProject インスタンスが 1 つ存在する．よって，プロジェクト名により，対象とするモデルを限定することが可能である[*1]．

2 に関して，各ノードに変換元インスタンスの継承情報は含まれていないため，抽出条件を満たしているかどうか，モデルだけでは判断できないという問題がある．

3 に関して，本プロパティグラフモデルにおいて，「A が B を含む」とは，「ノード A $\rightarrow$ ノード B」であることを意味している．つまり，2 で抽出したノードを始点として外向きに接続されているすべてのノードを抽出することになる．

(2)　局所的なデータの抽出（特定箇所データの部分抽出）

次に，局所的なデータ，例えば特定フロアのみのデータを抽出することを考える．IFC では，図 5.1 からも分かる通り，フロアなどの空間要素は階層構造で表現されており，IfcBuildingStorey インスタンスで表現する．図 5.1 には明示されていないが，IfcBuildingStorey インスタンスには，“GlobalID”, “Name”, “Description”, “Elevation” といった属性があり，これら属性を用いて複

[*1] プロジェクト名が一意に管理されていることを前提としている．もし，そうでない場合，GlolabID 等の別の一意な属性を用いれば良い．

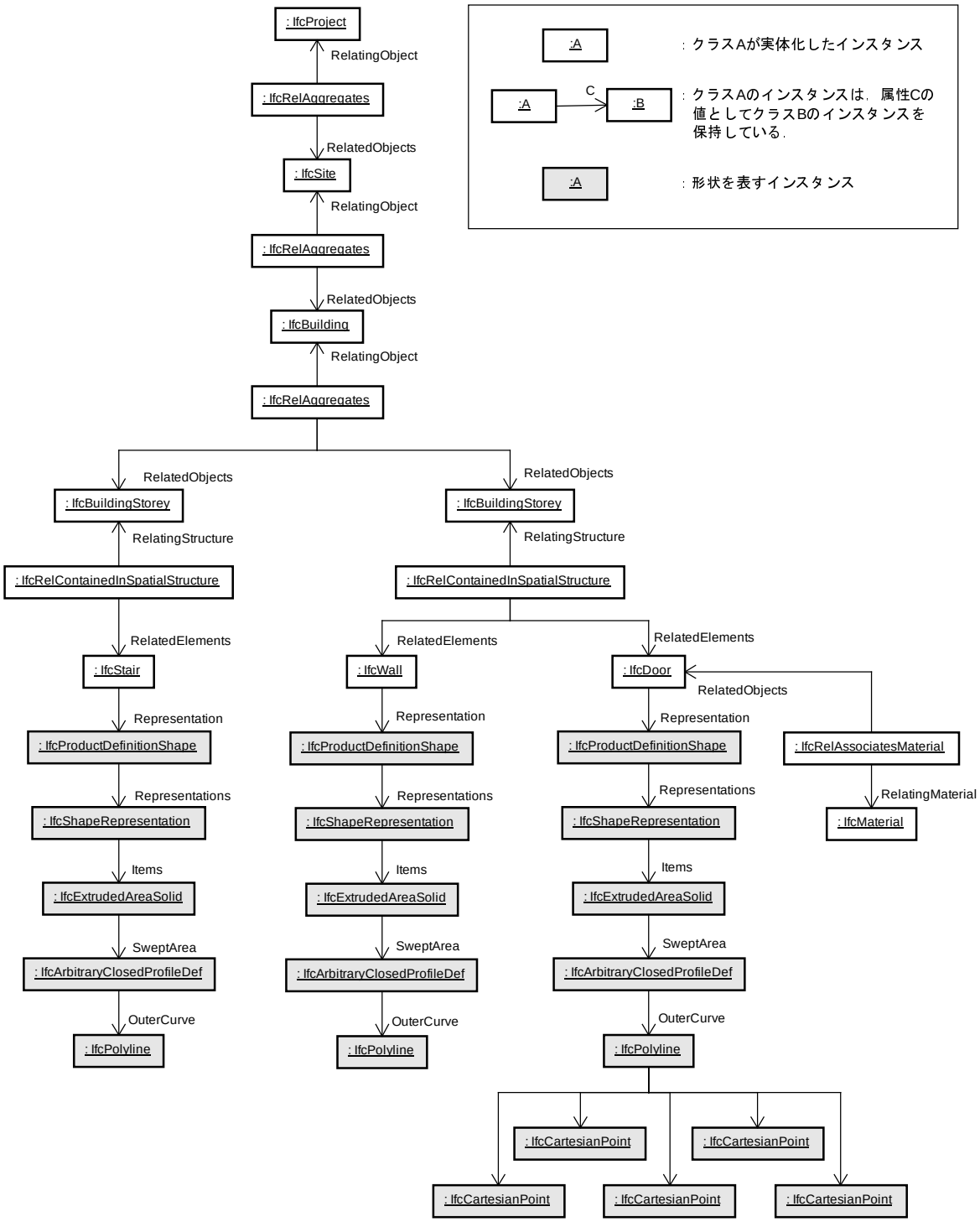

図 5.1　5F 建てビルディングのプロダクトモデルの一部（UML オブジェクト図）

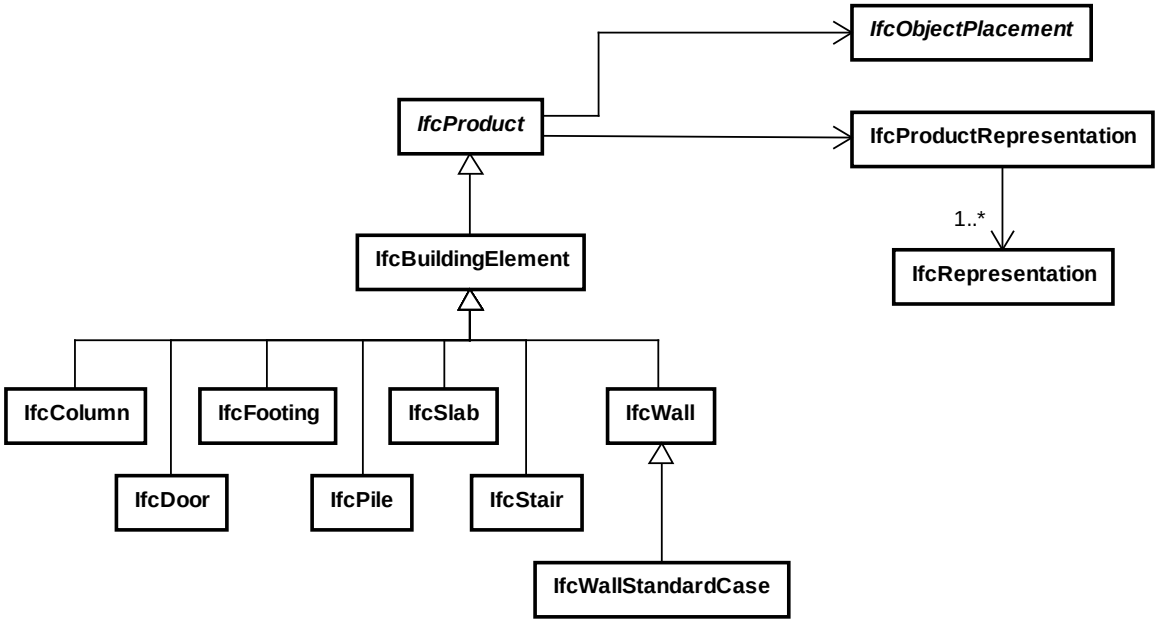

図 5.2　IfcProduct 関連クラス図

数フロアから，特定のフロアを選択することになる[*2]．対象のフロアが抽出されたら，そこに含まれるすべてのデータを抽出する．

これを整理すると以下になる．

1. 探索開始ノードをプロジェクト名により決定する．
2. 1 で決定したノードに接続されている全ノードのうち，IfcBuildingStorey インスタンスのノードで，かつ条件に合致するノードを抽出する．
3. 2 に含まれる全データを抽出する．

1 に関しては，「(1) 分散したデータの抽出」と同様である．

2 に関して，ノードのラベルにクラス名が設定されてあるため，特定種類のインスタンスのみ抽出することが可能である．さらに，インスタンスのプロパティの値を条件にして抽出するインスタンスを絞り込むことが可能である．

3 に関して，「(1) 分散したデータの抽出」と同様に，2 で抽出したノードを始点として外向きに接続されているすべてのノードを抽出すれば良いように思えるが，図 5.1 に示されている通り，空間要素である IfcBuildingStorey とそこに含まれる物理要素（IfcStair, IfcWall, IfcDoor 等）は IfcRelContainedSpatialStructure エンティティで逆向きに関係している（A ← IfcRelContainedSpatialStructure → B）.

[*2] IfcBuildingStorey に階数を示す属性は定義されていないため，しばしば Name 属性に階数が設定される．

5.2.2 プロパティグラフモデルの検討

第 4 章で開発した変換手法を用いて変換したプロパティグラフモデルから，5.2.1 項で示したアルゴリズムを用いて部分抽出することを考える．検討ポイントは以下になる．

- 各クラス間の継承情報の取得方法
- 関係性を示すオブジェクトの表現方法

第 4 章で開発した変換手法では，変換後のプロパティグラフモデルにクラス名は含まれるが，継承元クラス名は含まれないため，前述のアルゴリズムを適用することができない．また，要素同士の関係性を示すインスタンス（IfcRelationShip 派生クラスのインスタンス）が，プロパティグラフモデルでは外向きのエッジで他のノードと関連することになり，グラフの探索処理において計算量が大きくなる要因になる．

(1) 各クラス間の継承情報の取得方法

各クラス間の継承に関する情報は，IFC スキーマで定義されており，スキーマのバージョンが同じであれば継承情報も同じである．つまり，クラス間の継承情報は静的な情報であり，同一モデル内で動的に変化することはないため，システム内に 1 つ（または，バージョン毎に 1 つずつ）保持していれば良い．しかしながら，グラフデータベースは局所化したデータへのアクセス速度が高速である一方，広く分散したデータへのアクセス速度が低速であるため，対象プロダクトモデルの構成要素すべてに対して継承情報を取得する必要があるアルゴリズムでは，実用的な速度では動作しないと考えられる．

よって，データが冗長になってしまうが，各ノードに自ノード（インスタンス）の継承情報を含めることにした．第 3 章で述べたように，リレーショナルデータベースにおけるデータモデリング手法では，正規化によって冗長なデータを極力なくすことを目指す．冗長なデータを含むデータベースは，単に記憶領域が無駄であるだけでなく，冗長なデータの更新時に不整合が発生しやすいからである．しかしながら，このケースでは，継承情報は不変であり，変更されることはないため，記憶領域が無駄になるというデメリット以外は発生しない．

(2) 関係性を示すオブジェクトの表現方法

第 4 章の結果から分かるように，IFC プロダクトモデルの構造をそのままプロパティグラフモデルに変換した場合，IfcRelationship 派生インスタンスは関連するインスタンスを外向きのエッジで接続する．図 5.1 には以下の関係が存在するが，

$$\text{IfcBuildingStorey} \leftarrow \text{IfcRelContainedSpatialStructure} \rightarrow \text{IfcStair}$$

この関係は「あるフロア (IfcBuildingStorey) に，階段 (IfcStair) が含まれている．」ということを表現しており，IfcBuildingStorey から IfcStair への単一の向きの関係と解釈することが可能で

ある．

よって，第 4 章ではノードにより表現していた関係性の表現を，エッジによる表現に変更することにする．[*3]

5.2.3　プロパティグラフモデルへの変換手法の改良

第 4 章で示した変換手法に，前述した変更を加え，部分抽出に適した構造のプロパティグラフモデルへの変換手法を，以下の通りにまとめた．

1. 関係性を表すインスタンス以外のインスタンスに対応したノードを作成する．その際にノードのラベルにクラス名を格納する．
2. インスタンスの属性が単純データ型，列挙データ型である場合は，ノードに属性名と同名のプロパティを作成して属性値を格納する．
3. インスタンスの属性が集合体データ型であり，集合要素のデータ型が単純データ型，列挙データ型である場合は，ノードに属性名と同名のプロパティを作成して集合要素の配列を格納する．
4. インスタンスの属性がエンティティデータ型，あるいはエンティティデータ型の集合体データ型である場合は，属性に対応するインスタンスのノードを 1–3 の手順で作成し，作成したノードに向かう有効のエッジで接続する．さらに，クラスの継承情報を superClass プロパティに格納する．
5. 関係性を表すインスタンスの場合は，関連元インスタンスに対応するノードから関連先インスタンスに対応するノードに向かう有効のエッジで接続する．その際に，class プロパティに関係性を表すインスタンスのクラス名を格納する．

以上の規則で生成されるプロパティグラフモデルの例を図 5.3 と図 5.4 に示す．各インスタンス毎にクラスの継承情報を superClass プロパティに保持していることが分かる（図 5.3）．また，IfcRelationship 派生インスタンスがノードではなく，有向のエッジに変換されている（図 5.4 青線）．

5.2.4　クエリの作成手法

グラフ DBMS に格納されたプロパティグラフモデルから，実際に部分抽出処理を実行するには，クエリを作成する必要がある．本章で開発したアルゴリズムに対応したクエリの作成手法を以下に一般化する．

グラフデータベースのためのクエリは，リレーショナルデータベースにおける SQL のように標

[*3] グラフのエッジを関係性と解釈することは，多くのグラフデータベースアプリケーションで行なわれている。よく知られているものにソーシャルグラフの生成がある。ソーシャルグラフとは，ソーシャルメディアにおける，人と人，または組織同士の繋りを表現するためのもので，グラフデータベースを利用することにより，詳細な関係性を表現することが可能になっている．

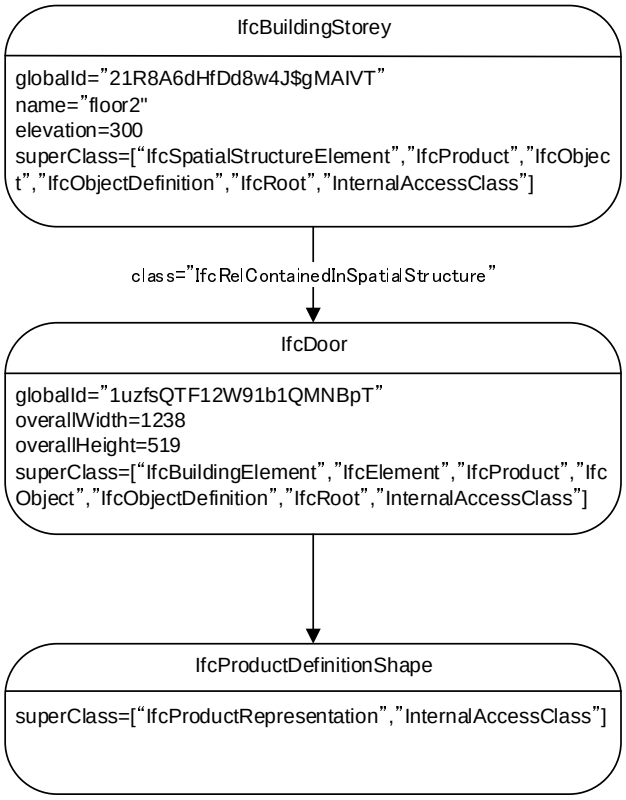

図 5.3 IfcRelationship によるプロパティグラフの表現

準化されたものが存在しない．本章では，Neo Technology 社の Neo4j [1] という DBMS 製品のクエリ言語である，Cypher [2] と呼ばれるクエリ言語を利用する．

(1) 分散したデータの抽出（全形状データの抽出）

1. 探索開始ノードをプロジェクト名により決定する．
   プロジェクト名を引数として，以下クエリに一般化できる．

```
MATCH (p:IfcProject) WHERE p.name="プロジェクト名" WITH p
```

2. 1 で決定したノードに接続されている全ノードのうち，指定クラス派生インスタンスのノードを抽出する．

```
MATCH (p:IfcProject)-[*]->(d) WHERE "IfcProduct" IN d.superClass
 WITH d
```

3. 2 に含まれる形状データ（形状種類，および位置情報）を抽出する．

```
MATCH (d)-[*]->(f:IfcFace)-[*]->(:IfcPolyLoop)-->
(c:IfcCartesianPoint)
```

4. 形状データを返す．

```
RETURN f, c;
```

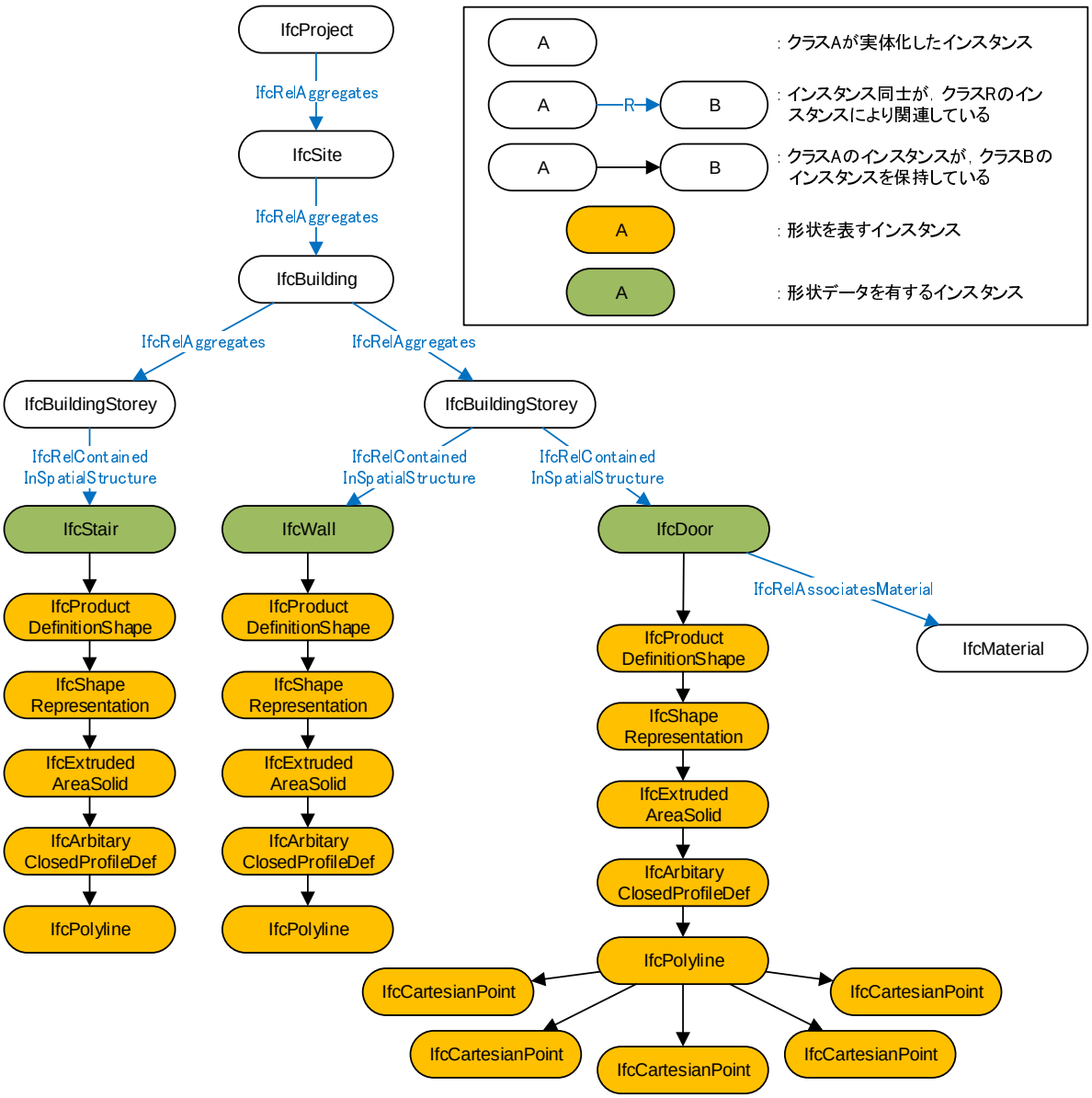

図 5.4　IfcRelationship エッジによるプロパティグラフの表現

以上の手順で得られたクエリを結合して 1 つのクエリとする．

(2)　局所的なデータの抽出（特定箇所データの部分抽出）

1. 探索開始ノードをプロジェクト名により決定する．
   プロジェクト名を引数として，以下クエリに一般化できる．

   ```
   MATCH (p:IfcProject) WHERE p.name="プロジェクト名" WITH p
   ```

2. 1 で決定したノードに接続されている全ノードのうち，IfcBuildingStorey インスタンスのノードで，かつ条件に合致するノードを抽出する．

```
MATCH (p)-[*]->(f:IfcBuildingStorey) WHERE f.name="条件" WITH f
```

3. 2 に含まれる全データを抽出する.

```
MATCH (f)-[*]->(n)
```

4. 2 に含まれる全データを返す.

```
RETURN f, n;
```

以上の手順で得られたクエリを結合して 1 つのクエリとする.

5.3 検証

考案した変換手法と部分抽出手法が，実際のプロダクトモデルに適用可能であることを検証する．検証対象データとして 5 階建ビルディングの IFC プロダクトモデル (図 5.5，表 5.1) を用いた．まず，Java 言語で作成したプログラムを用いて IFC プロダクトモデルをプロパティグラフモデルに変換し，グラフ型 DBMS に格納する．その後，クエリ言語を用いて特定箇所の形状データを抽出する.

検証環境を表 5.2 に示す.

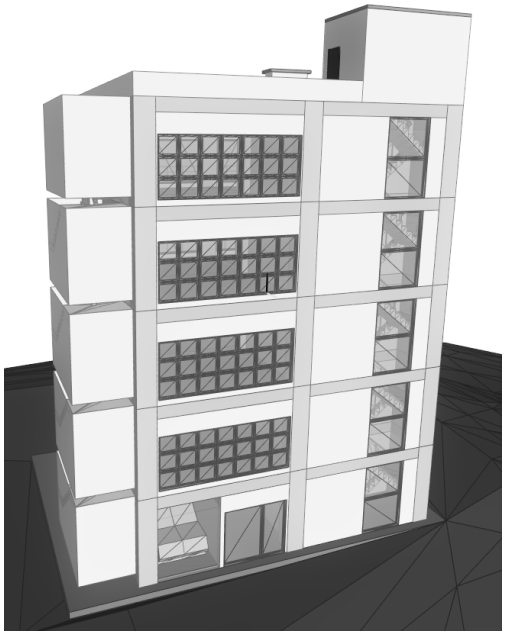

図 5.5　検証対象プロダクトモデルの表示イメージ

表 5.1　検証対象プロダクトモデルの情報

項目	数量
ファイルサイズ	51Mバイト
全インスタンス	1,003,212
最大階層数	19
平均階層数	11

表 5.2　検証環境

項目	スペック等
CPU	Intel Core i5-2520M（2.5GHz, 2コア）
メモリ	16GB (DDR3 133MHz)
ストレージ	240GB SSD
OS	Mac OS X 10.12 64bit
DBMS	Neo4j 3.1.4

表 5.3　抽出対象箇所（階段）のインスタンス数

要素	クラス	個数
階段本体	IfcStair	1
部品	IfcOpenShell	62
ポリゴン（面）	IfcPolyLoop	13,440
頂点	IfcCartesianPoint	40,320

5.3.1　検証方法

検証対象データである 5F 建てビルディングのプロダクトモデルから，5F 部分の階段の形状データを取得する．この階段の形状は 3 角ポリゴンの集合で構成されており，Neo4j に格納する前の段階（Part21 形式のファイルの状態）で形状を構成する要素の個数を予め計測しておく (表 5.3)．要素の計測には IFCExplorer[*4] というツールを用いた．

図 5.6 のクエリを実行すると，階段の形状データ（面と点）が抽出される．実際には，面 (IfcPolyLoop インスタンス）の ID と面を構成する点（IfcCartesianPoint インスタンス）の座標値がコンマで区切られたテキスト形式（CSV 形式）で出力される．この抽出結果から形状データの面と点の個数を数え，元データと合致していることを確認する．また，抽出した座標値から，obj 形式 [3] のファイルを作成し，ビューアで形状を確認する．

[*4] buildingSMART Japan が開発した IFC プロダクトモデルの解析ツールである．クラス種別による検索や，インスタンスのツリー表示等様々な機能を搭載している．

```
1 MATCH (p:IfcProject) WHERE p.name="LUC-01" WITH p
2 MATCH (p)-[*]->(f:IfcBuildingStorey) WHERE f.name="Mod-Floor-5"
  WITH f
3 MATCH (f)-[*]->(s) WHERE "IfcProduct" IN s.superClass AND s.
  shapeType="STAIR" WITH s
4 MATCH (s)-[*]->(l:IfcPolyLoop)-[r:AGGREGATES]->(p:
  IfcCartesianPoint)
5  RETURN id(l), p.coordinates[0], p.coordinates[1], p.coordinates
  [2]
6  ORDER BY id(l), r.index
```

図 5.6　階段形状抽出クエリ

表 5.4　抽出対象箇所（階段）のインスタンス数

要素	クラス	個数
ポリゴン（面）	IfcPolyLoop	13,440
頂点数	IfcCartesianPoint	40,320

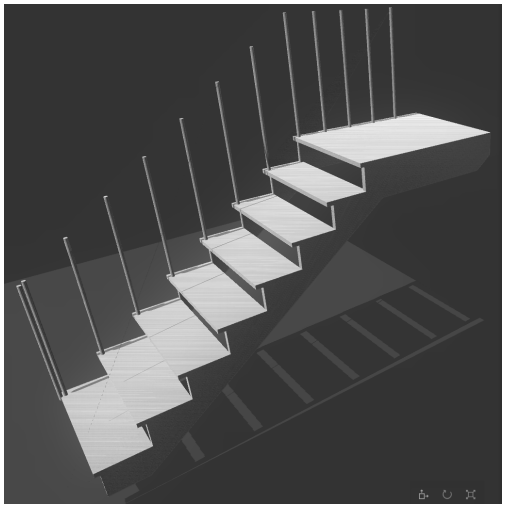

図 5.7　抽出結果の表示イメージ

5.3.2　結果

図 5.6 のクエリにより抽出した形状データの情報を表 5.4 に示す．また，抽出した頂点データを obj 形式に変換後のファイルをビューアで表示した結果を図 5.7 に示す．正しく抽出されていることが分かる．なお，CSV 形式の頂点データから obj 形式への変換は，Java 言語で簡易的な変換プログラム[*5]（付録 A.1）を作成して変換した．

[*5] IFC には様々な形状が定義されているが，このプログラムでは IfcFacetedBrep にしか対応していない．

5.4　考察

実行したクエリの意味であるが，最初の 3 行は，Neo4j DBMS に格納されている様々な建築物データから，抽出する建築物とそのフロアを特定し，さらに形状を含むインスタンスのみ抽出することを示している．プロジェクト (建築物全体を示す) 名称”LUC-01”とフロア名称”Mod-Floor-5”により，ビルディングの 5F までを特定し，さらに IfcProduct の派生クラスのインスタンスのみ抽出することで，形状を含まないインスタンスを除外している．次の 2 行は，抽出する形状データを特定部材（階段）に特定し，それを構成する面と点を抽出することを意味している．最後の行は，出力データを示しており，面を識別するための ID と，各頂点の座標値 (x, y, z) を出力している．

IFC プロダクトモデルが階層構造となっていることは，図 5.1 に示したが，これは上位の要素が下位の要素を含む構造となっているため，最上位の要素であるプロジェクト (IfcProject) から，下位の要素をたどって行くことで，末端の座標点 (IfcCartesianPoint) までを取得できる．つまり，IFC プロダクトモデルにおける抽出処理は，①全建築物，②特定建築物，③特定フロア，④特定種類部材，⑤特定種類データの順に，対象データを絞り込んでいく処理となる．

グラフデータベースにおける抽出（検索）処理は，対象となる全ノードを走査していく処理であり，対象ノード数を n とすると，通常 $O(n)$ で処理速度が増加する．よって，様々な建築物の情報を集計するような処理は苦手としているが，上記のような対象ノードを絞り込んでいくような処理では，グラフ走査処理が局所化するため高速に動作する．

つまり，部分抽出処理において，IFC プロダクトモデルのデータ構造は，グラフデータベースに対して適合性が高い．また，IFC プロダクトモデルのこのような構造は，IFC スキーマのバージョンに依らない基本的な構造であるため，IFC4 など他のバージョンのスキーマで記述されたプロダクトモデルに対しても同様の結果が得られる．

5.5　本章のまとめ

グラフ DBMS で採用されているプロパティグラフモデルは，構成要素同士の関係性を詳細に記述する IFC スキーマの設計方針と相性が良く，JSON 等のオブジェクト指向をベースとしたモデリング言語を用いるよりも簡潔にモデル化可能であることを示した．

さらに，IFC プロダクトモデルをグラフ型 DBMS に格納するための変換手法を考案し，変換後のモデルを格納した DBMS から特定のデータのみを，グラフ型 DBMS に備わっている機能（クエリ）だけを用いて，複雑なプログラムを作成する必要なく，簡単に抽出可能であることを検証した．この事実は，ネットワーク上でのプロダクトモデルの共有をシステム化する際に，グラフ型 DBMS を用いることでシステム化が容易になる可能性が高いことを示唆している．

本研究では，グラフデータベースを使用した部分抽出処理手法そのものに着目したため，その処理速度については考察しなかった．実証した抽出処理は実用範囲内の速度 (4,242ms) で動作したが，十分に速いとは言えない．また，実際にシステム化するためには，抽出処理だけでなく，追加，

変更，削除，さらに複雑な検索といった処理も必要になる．よって，抽出処理の高速化，および，グラフデータベースにおける，追加，変更，削除処理の検証，さらに，実業務における利用を考慮した検索処理手法の確立が，今後の課題となる．

参考文献

[1] Neo Technology, Inc.: Other Neo4j Releases,
https://neo4j.com/download/other-releases/ (参照 2017/6/1).

[2] Neo Technology, Inc.: The Neo4j Developer Manual v3.1, 2016,
https://neo4j.com/docs/developer-manual/3.1/ (参照 2017/6/1).

[3] ウィキペディア, Wavefront .obj ファイル, 2016, https://ja.wikipedia.org/wiki/Wavefront_.obj%E3%83%95%E3%82%A1%E3%82%A4%E3%83%AB (参照 2017/6/1).

第 6 章

グラフ DBMS を用いた BIM/CIM 情報共有システムの提案

6.1　本章の概要

第 1 章において，BIM/CIM 情報共有システムの必要性を述べた．そして，第 3 章から第 5 章で，BIM/CIM 情報共有システムの構築に必要なデータベース技術について検討し，性能面において，現在広く用いられているリレーショナルデータベースよりも，グラフデータベースにメリットが大きいことを示した．

本章では，実際に BIM/CIM 情報共有システムを構築することを想定して，グラフ DBMS を用いたシステムのメリット，またはデメリットを性能面に関してだけでなく，開発コスト，メンテナンスコストの視点からも検討する．その結果，総合的にグラフデータベースを用いたシステムの優位性が明らかになると考えている．

6.2　情報共有システムを中心とした BIM/CIM の姿

6.2.1　現在の情報共有システム

日本国内では，BIM/CIM が始まる以前より，工事情報共有システムを利用した（主に）施工時における関係者間の情報共有が行なわれている．そこで，詳細な検討に入る前にそれらシステムの概要を示す．

国土交通省は，公共事業分野の生産性の向上を目的として，CALS/EC（公共事業支援統合情報システム）の構築を推進し [1]，その一環として，「工事施行中における受発注者間の情報共有システム機能要件」を発行した [2]．この機能要件は，その名の通り，工事施工中において受注者と発注者間における情報共有を実現するシステムの基本機能が定義されている．国土交通省がシステムを構築，運営するのではなく，民間のシステムベンダーが本機能要件に合致したシステムを開発，

運営し，公共工事の期間中に ASP (Application Service Provider）方式[*1]で運営されるのが特徴である．なお，機能要件は順次改訂され現在は Rev.4.0 となっている．

国土交通省が提供している情報によれば，機能要件 Rev.4.0 に対応している工事情報共有システムは 9 つ存在しており [3]，国土交通省発注の土木工事のほぼすべてでそれらが使われている．同様に，地方自治体においても国土交通省の機能要件に準じた工事情報共有システムの利用が広がっている [4–6].

一方，民間発注の工事においても，大規模な工事であれば，国土交通省の工事情報共有システムと類似したシステムで情報共有を行なっている．しかし，国交省の情報共有システムと異なり，標準は存在せず，工事を請負ったゼネコンなどが独自でシステムを調達している [7].

6.2.2　BIM/CIM 情報共有システム

第 1 章で，BIM/CIM が成熟していくに従って，ファイルとローカルデータベースによる情報共有から，ネットワーク上で統合されたシステムに進化していくと考えられていると述べた．上述したように，BIM/CIM が主に対象とする比較的大規模な工事では，すでに工事情報共有システムが導入されているため，日本国内においては，図 6.1 に示すように，すでに導入済の工事情報共有システムを情報基盤として BIM/CIM が進化していくと考えるのが自然である．システムが 3 次元プロダクトモデルを扱えるようになり，システムの利用が工事施行中の発注者（施主）と受注者（施工者）間に限られていたのが，多数の関係者間に広がっていることが分かる．

6.2.3　システム構成

(1)　一般的な工事情報共有システムの構成

国土交通省の「工事施行中における受発注者間の情報共有システム機能要件 (Rev.4.0)」に対応した工事情報共有システム[*2]の 1 つである basepage [8] のシステム構成の概要を図 6.2 に示す．リレーショナル DBMS を用いた Web3 層アーキテクチャであることが分かる．

前述した通り，公共工事における工事情報共有システムは，国土交通省の機能要件に基いて開発されているが，この機能要件に示されているのは要求仕様[*3]と非機能要求[*4]が中心であり，システム構成については規定されていない．また，機能要件に対応した 9 つのシステムはすべて商用システムであり，内部のシステム構成については公開されていないため，それらすべてのシステム構成を知ることはできないが，図 6.2 に示した構成は，一般的な Web システムでよく用いられる構成であり，数多くの事例と長い運用実績があるため，その他の情報共有システムにおいても同様の構

[*1] 商用システムの提供形態の 1 つで，利用者はシステムを購入するのではなく，決められた期間借り受ける方式である．

[*2] BIM/CIM に対応していない現状の工事情報共有システムを，単に “工事情報共有システム” と呼び，BIM/CIM に対応した工事情報共有システムを “BIM/CIM 情報共有システム” と呼ぶ．

[*3] システムに対する機能に関する要求のことである．

[*4] システムに対する機能以外の要求であり，性能面やセキュリティ面に関する要求がこれにあたる．

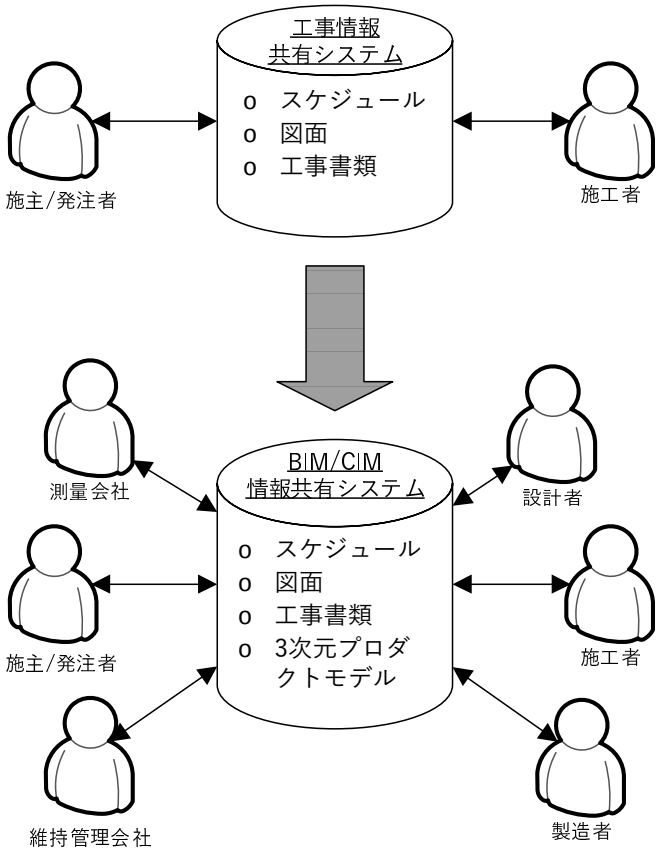

図 6.1　予想される情報共有システムの進化

成であると考えられる．

よって，本章では，現状の情報共有システムの構成が図 6.2 と同等であることを前提に検討を行なうことにする．

(2)　提案する BIM/CIM 情報共有システムの構成

第 4 章で，BIM/CIM において中心的な存在である 3 次元プロダクトモデルが，リレーショナル DBMS が採用するデータモデルである関係モデルと親和性が低く，性能が発揮できないことを明らかにした．しかしながら，現在の一般的な情報共有システムはリレーショナル DBMS を用いて構築されており，システム構成を変えずに，単に機能を拡張するだけでは，特に性能面で問題が発生すると予想される．

よって，本章では本論を通じて研究の対象としているグラフ型 DBMS を用いた BIM/CIM 情報共有システムを提案する．そのシステム構成を図 6.3 に示す．データベースサーバがリレーショナル型とグラフ型を併用する構成となっているのが特徴である．

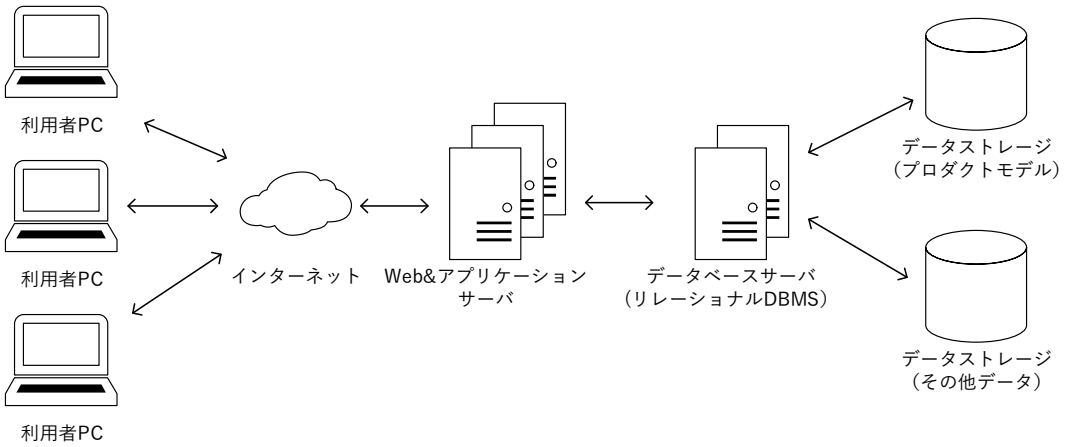

図 6.2　一般的な工事情報共有システムの構成

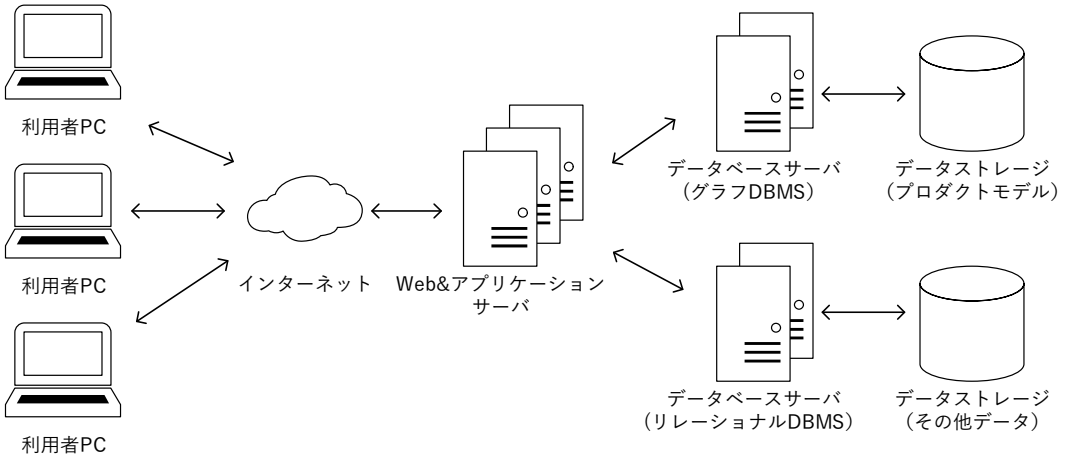

図 6.3　提案する BIM/CIM 情報共有システムの構成

6.3　コストに関する検討

BIM/CIM 情報共有システムを開発することを想定して，リレーショナル DBMS を用いた場合と，グラフ DBMS を用いた場合の開発コストの違いを考察する．BIM/CIM 情報共有システムに求められる機能は多岐に渡るため，すべての機能に関して考察することは困難である．そこで，本研究ではプロダクトモデル読み込み処理の一部，入力データをそれぞれのデータモデル，関係モデルとプロパティグラフモデルに変換して DBMS に格納する処理を考察対象とした．理由は，データ処理部分に DBMS の違いが現れやすいからである．

コストの見積りには，COCOMO モデル [9] を利用した．

6.3.1 COCOMO モデル

COCOMO は，開発規模から開発期間と開発にかかるコスト（人月）を算出するためのモデルである．扱うデータの複雑度と処理の複雑度から開発規模を算出するファンクションポイント法 [10] と並び，ソフトウェア開発の見積りに広く用いられている．

$$PM = a(KDSI)^{E1}\Pi A \tag{6.1}$$
$$TDEV = b(PM)^{E2} \tag{6.2}$$

PM : 人月
$TDEV$: 標準開発期間（月）
$KDSI$: 開発されるプログラム命令数（k ステップ）
ΠA : 努力係数
a, b : 定数
$E1, E2$: 乗数

式 6.1 が，開発規模 (KDSI) から開発コスト (PM) を算出する式であり，式 6.2 が，開発コスト (PM) から開発期間 (TDEV) を算出する式である．入力値である開発規模はステップ数[*5]で与える必要があるが，通常，ソフトウェアの見積り時にプログラムコードは存在していないため，ファンクションポイント法で開発規模を別途見積った後，ステップ数に変換する手法がよく用いられている [11].

努力係数 ΠA，定数 a, b，乗数 $E1, E2$ は，開発プロジェクトの規模や，開発者の技量，時間的な制約等の様々な要因を総合的に考慮して決定される（表 6.1, 表 6.2）.

6.3.2 開発コストの比較検討

COCOMO を用いて，リレーショナル DBMS のみを用いた構成とグラフ DBMS を並用する構成の，開発コストの比較検討を行なった．

(1) 努力係数 ΠA の決定

COCOMO を用いた見積りでは努力係数を決める必要があるが，COCOMO では表 6.1 にあるように 15 個の努力係数が定義されており，それぞれを掛け合わせて全体の努力係数 ΠA とする．本章の目的の 1 つは，BIM/CIM 情報共有システムという機能が同一であるシステムにおいて，システムを構成する DBMS の違いが開発コストにどのように影響するのか明らかにすることであ

[*5] プログラムの規模を示す値であり，プログラムコードを算出規則に従って数えることで算出する．複数の算出規則が提案されているが標準は存在しない．

表 6.1　COCOMO の努力係数（出典:プロジェクトコスト見積り入門 [11]）

属性	努力係数	定義	レベル					
			非常に低い	低い	中位	高い	非常に高い	極めて高い
ソフト製品	RELY	ソフトウェアに要求される信頼性	0.75	0.88	1.00	1.15	1.40	—
	DATA	データベースのサイズ	—	0.94	1.00	1.08	1.16	—
	CPLX	ソフトウェア製品の複雑性	0.70	0.85	1.00	1.15	1.30	1.65
ハード製品	TIME	システムに課せられた実行時間制約	—	—	1.00	1.11	1.30	1.66
	STOR	主記憶装置制約	—	—	1.00	1.06	1.21	1.56
	VIRT	OS・ハードウェアの変更履歴	—	0.87	1.00	1.15	1.30	
	TURN	要求されるターンアラウンドタイム	—	0.87	1.00	1.07	1.15	
要因	ACAP	アナリスト能力	1.46	1.19	1.00	0.86	0.71	—
	AEXP	同様のアプリケーションの経験度合い	1.29	1.13	1.00	0.91	0.82	—
	PCAP	プログラマ能力	1.42	1.17	1.00	0.86	0.70	—
	VEXP	OS/ハードウェアの経験度合い	1.21	1.10	1.00	0.90	—	—
	LEXP	プログラミング言語の経験度合い	1.14	1.07	1.00	0.95	—	—
プロジェクト	MODP	ソフトウェア開発技法の使用度合い	1.24	1.10	1.00	0.91	0.82	—
	TOOL	ソフトウェア・ツールの使用度合い	1.24	1.10	1.00	0.91	0.83	—
	SCED	要求される開発スケジュールの制約	1.23	1.08	1.00	1.04	1.10	—

表 6.2　COCOMO の開発モード（出典:プロジェクトコスト見積り入門 [11] より作成）

	組織モード	半組込モード	組込モード
製品目標の組織的理解	・十分に理解	・かなり理解	・一般的な理解
同様のソフトウェア開発経験	・広く経験	・かなり経験	・適度の経験
最新技術の必要性	・最小限	・いくつかは必要	・かなり必要
開発規模	・50KSLOC未満	・300KSLOC未満	・300KSLOC以上
対象システム	・簡単な小型システム	・多くの業務処理システム ・ 多くのリアルタイム処理	・大規模な業務システム ・超大型リアルタイム処理 ・大規模OS
定数	a=3.2, b=2.5	a=3.0, b=2.5	a=2.8, b=2.5
乗数	$E1$=1.05, $E2$=0.38	$E1$=1.12, $E2$=0.35	$E1$=1.20, $E2$=0.32

る．つまり，比較により違いが明らかになればよく，正確な開発コストを求める必要はない．よって，両システム構成で共通の係数はすべて中位 (1.0) とした[*6]．

両システム構成で異なる係数は，MODP（ソフトウェア開発技法の使用度合い）のみと考えられる．リレーショナル DBMS のみを用いるシステム構成は前述した通り一般的であり，使用度合い（習熟度）が高いため非常に高い (0.82) とし，一方のグラフ DBMS を並用するシステムでは，少なくとも現状では一般的ではないため，非常に低い (1.24) とした．他はすべて中位 (1.0) である

[*6] COCOMO で正確な開発コストを求めるには，開発実績に基いた算出工数の調整が不可欠であり，詳細な検討により係数を中位 1.0 以外に設定したとしても，正確な工数が算出できるわけではない．

ため，全体の努力係数 ΠA はそれぞれ，0.82 と 1.24 になる．

(2) 定数と乗数の決定

COCOMO を用いた見積りでは定数 a, b と乗数 $E1, E2$ を決める必要がある．これらは，表 6.2 に示した開発モード（プロジェクト全体の難易度を表わす）によって決まっている．BIM/CIM 情報共有システムは，簡単な小型システムではないことは間違いないが，大規模 OS に匹敵するほど大規模ではないため，半組込モードを選択した．

よって，両システム構成共に，$a = 3.0, b = 2.5, E1 = 1.12, E2 = 0.35$ となる．

(3) 開発規模の算出

COCOMO による開発コストの見積りには，対象プログラムの開発規模が必要であり，前述したように，一般的にはファンクションポイント法などを用いて概算規模を算出するが，本章では，より正確な開発規模を算出するため，それぞれの DBMS 用のプログラムコードを一部作成し，そのコードを元に全体の開発規模を算出する．

データベースを利用したシステムの開発では，DAO (Data Access Object) を用いた設計が一般的である．Alur らによってデザインパターンとしてまとめられた結果，システム開発で広く用いられる設計パターンとなっているため [12]，本論でもこの設計にならうことにする．データ処理部分のクラス構成を図 6.4 に示す．IFC スキーマの各エンティティに対応するクラスが Data クラスであり，DataAccessObject クラスが DBMS（DataSource クラス）にアクセスし，得た情報を元に Data オブジェクトを生成している．

そこで，ある IFC4 スキーマのエンティティに対して，リレーショナル DBMS 用とグラフ DBMS 用の DataAccessObject クラスを作成（Java 言語によりプログラミング）し，両者のステップ数を計測する．そして，その結果から全体の開発規模を算出することにする．具体的には，鉄筋を表す IfcReinforcingBar エンティティを選択した．実際にプログラミングの対象となるクラ

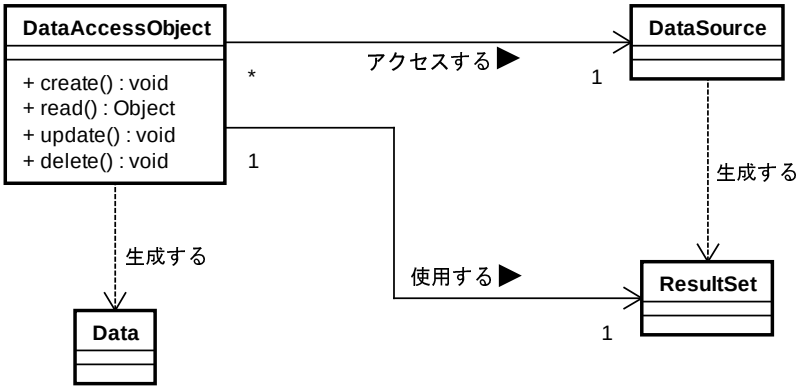

図 6.4 Data Access Object のクラス図（出典:J2EE パターン [12]）

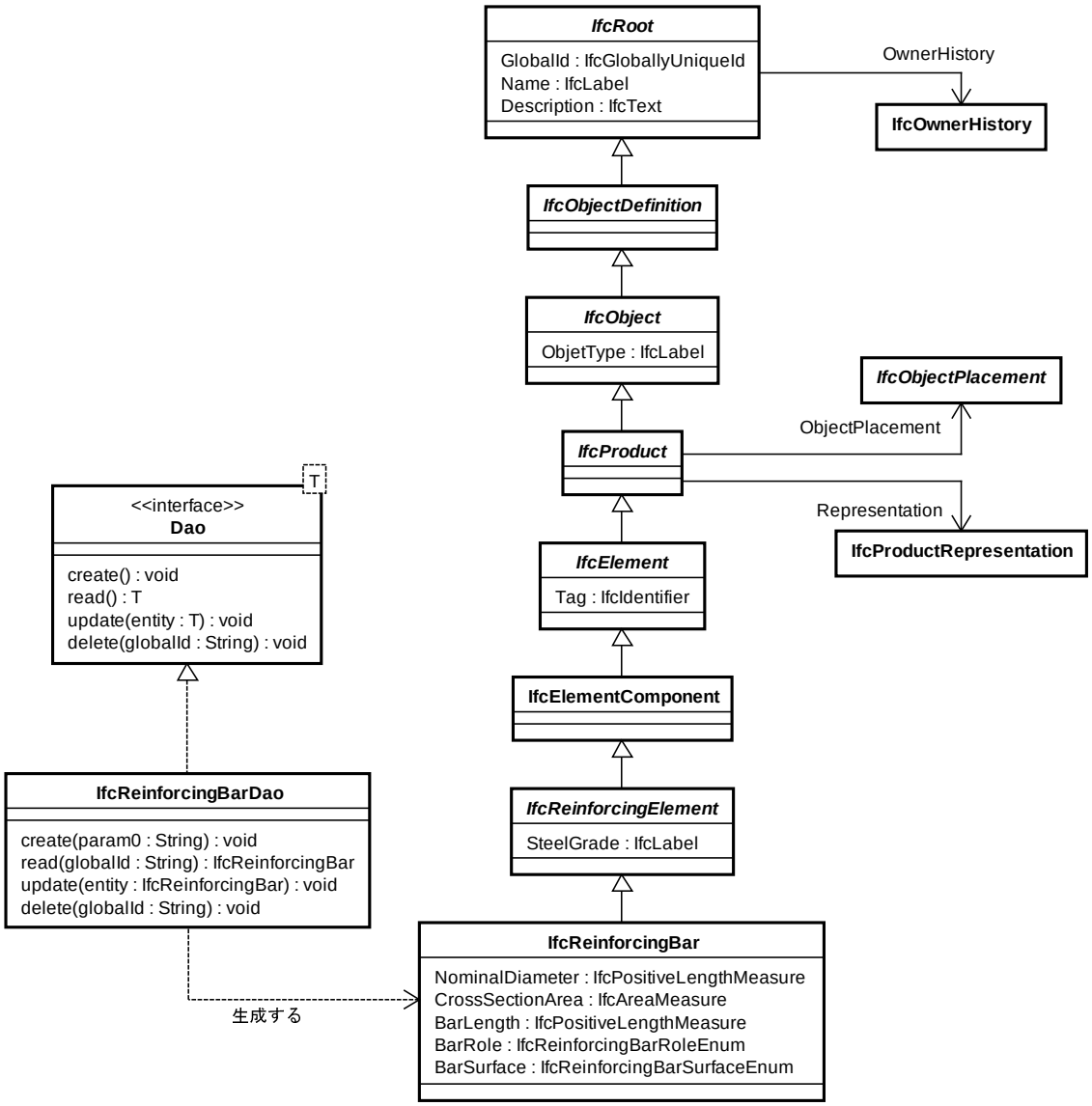

図 6.5　IfcReinforcingBar と Data Access Object（UML クラス図）

スを図 6.5 に示す．これを元にプログラムコード（付録 A.3, A.4）を記述し，ステップ数[*7]を計測した．計測対象のプログラムコードは，IfcReinforcingBarDao のみとし，エンティティを表わすクラスである IfcRoot や IfcProduct，IfcReinforcingBar 等は含めていない．なぜなら，エンティティを表わすクラスは商用ライブラリを用いるのが一般的だと考えたのと，エンティティに関わるプログラムコードはリレーショナル DBMS 用とグラフ DBMS 用とで同一であり，比較検討に関係しないためである．

[*7] 複数の数え方が提唱されているが標準は存在しないため，本論ではプログラムコードからコメント行と空行を除いた行数である SLOC (Source Lines Of Codes) をステップ数として用いた．また，ステップ数の計測には SLOCCount というツール (`https://www.dwheeler.com/sloccount/`) を利用した．

計測した IfcReinforcingBarDao のステップ数から，全体の開発規模を算出した結果が以下である．

1. リレーショナル DBMS を用いたシステムの開発規模

$$224 \times 543/1000 = 121.6$$

2. グラフ DBMS を用いたシステムの開発規模

$$1010/1000 = 1.01$$

リレーショナル DBMS 用の DAO プログラムコードは，IFC4 スキーマに定義されているインスタンス化可能なエンティティ 543 個分[*8]のコードが必要になるが，グラフ DBMS 用のコードは，1 種類で済むため，開発規模が非常に小さい．

(4) 開発コストの算出

最後に，(1)〜(3) で求めた，努力係数，定数，乗数，開発規模から，COCOMO を用いて開発コストを算出する．結果は以下の通りである．

1. リレーショナル DBMS を用いたシステムの開発コスト

$$PM = 3.0 \times (121.6)^{1.12} \times 0.82 = 532.2\ (\text{人月})$$
$$TDEV = 2.5 \times (PM)^{0.35} = 22.5\ (\text{ヶ月})$$

2. グラフ DBMS を用いたシステムの開発コスト

$$PM = 3.0 \times (1.01)^{1.12} \times 1.24 = 3.8\ (\text{人月})$$
$$TDEV = 2.5 \times (PM)^{0.35} = 2.2\ (\text{ヶ月})$$

6.3.3 開発コストに関する考察

前節での開発コストの算出では，システム構成の違いにより大幅な差異（140 倍）が出た．この理由について，以下で考察する．

(1) リレーショナル DBMS を用いた構成の開発コスト

図 6.4 の設計では，IFC のエンティティ毎にデータアクセス用のクラスを定義することになる．リレーショナル DBMS はスキーマ必須の DBMS であるため，データの格納先となるテーブルを予め定義しておく必要がある．具体的にはテーブルの名称と，そのテーブルが含むカラムの個数，名称，データ型等である．プログラム側で IFC インスタンスを DBMS に登録する場合，各インス

[*8] IFC4 に定義済の全エンティティ 766 個から，インスタンス化できない ABSTRACT であるエンティティ 123 個を除いた個数.

タンスを関係モデルに変換した後，インスタンスの種類毎に格納先のテーブルを振り分けて格納する必要がある．よって，リレーショナル DBMS を用いたシステムの開発では，IFC スキーマで定義されている 543 個のエンティティに対応するテーブルを予め定義し，それらのテーブルに 1 対 1 に対応するデータアクセスクラスをプログラミングする必要がある．さらに，IFC スキーマのバージョンアップが発生したり，異なる IFC 拡張スキーマへの対応が必要になった場合には，対応するスキーマ毎に同様の対応が必要になる．

(2)　グラフ DBMS を用いた構成の開発コスト

一方，グラフ DBMS はスキーマレスの DBMS であるため，格納するデータの形式は，プロパティグラフモデルの形式に従っている限り自由である．そのため，プログラムは IFC インスタンスをプロパティグラフモデルに変換した後，そのままグラフ DBMS に追加すれば良い．追加するデータ（ノードまたはエッジ）の名前，そこに含まれる属性の個数，名称，データ型を予め決めておく必要はない．よって，グラフ DBMS を用いた BIM/CIM 情報共有システムの開発，メンテナンスでは，リレーショナル DBMS を用いたシステムの開発で発生する，データベーススキーマの定義，データアクセスクラスのプログラミングが不要になる．つまり，システム開発コスト，メンテナンスコストが大幅に低い．

6.4　まとめ

第 4 章で行なった性能比較では，リレーショナルデータベースとグラフデータベースのデータモデルの違いが性能の違いになった．本章における，開発コストの比較検討では，各データベースのスキーマの定義の方法，つまり，固定スキーマであるかスキーマレスであるか，の違いが結果に大きく作用した．第 3 章で述べたように，NoSQL に分類されるデータベースは，ほとんどがスキーマレスのデータベースであり，IFC のように多数のエンティティが定義されている複雑なプロダクトモデルに対してシステム化するには，スキーマレスのデータベースが適していると言える．

参考文献

[1] 日本建設情報総合センター: CALS/EC とは？ CALS/EC イントロダクション, http://www.cals.jacic.or.jp/calsec/index.html (参照 2017/6/1).

[2] 国土交通省: 工事施工中における受発注者間の情報共有システム機能要件（Rev.4.0）【要件編】, 2014.

[3] 国土交通省: 情報共有システム提供者における機能要件 (Rev4.0) 対応状況一覧表, 2017, http://www.cals-ed.go.jp/mg/wp-content/uploads/soukatsuichiran4.pdf (参照 2017/6/1).

[4] 兵庫県県土整備部県土企画局契約管理課: CALS/EC について, 2016, https://web.pref.hyogo.lg.jp/ks03/wd04_000000002.html (参照 2017/6/1).

[5] 広島県土木建築局技術企画課: 広島県工事中情報共有システム, https://chotatsu.pref.hiroshima.lg.jp/asp/index.html (参照 2017/6/1).

[6] 岡山県土木部技術管理課管理情報班: 岡山県公共工事施工管理支援(情報共有)システム, http://www.pref.okayama.jp/doboku/gikan/cals/cals_johos.html (参照 2017/6/1).

[7] サイボウズ: サイボウズ Office6 導入事例, https://cybozu.co.jp/casestudy/example/taisei_kensetsu/pdf/taisei_kensetsu.pdf.

[8] 川田テクノシステム: 情報共有 ASP basepage, 2017, http://www.kts.co.jp/asp/basepage/index.html (参照 2017/6/1).

[9] Center for Software Engineering, USC: COCOMO II Model Definition Manual Version 2.1, 2000.

[10] International Function Point Users Group (IFPUG), 日本ファンクションポイントユーザ会 (JFPUG) 訳: ファンクションポイント計測マニュアル (CPM:Counting Practice Manual) リリース 4.1.1, 2001.

[11] 岡村正司: プロジェクトコスト見積り入門, pp.123–128, 日経 BP 社, 2009.

[12] Deepak Alur, John Crupi, Dan Malks, ウルシステムズ（監訳）, 近棟稔（監訳）, 吉田悦万（監訳）, 小森美智子（監訳）, トップスタジオ（訳）: J2EE パターン第 2 版, pp.458, 日経 BP 社, 2005.

[13] 建設情報標準化委員会, 電子成果高度利用検討小委員会: 工事施工中における受発注者間の情報共有「情報共有のあるべき姿」（案）, 2006.

第 7 章

総括

7.1 結論

本研究は，BIM/CIM における複数の関係者による情報共有と協調作業を実現するために，最も重要な技術の 1 つであるデータマネジメント技術に着目し，プロダクトモデルの DBMS への格納手法と部分抽出手法を開発し，さらに，実際にシステム化することを想定して，開発コストとメンテナンスコストについて検討したものである．現在，圧倒的なシェアを占めるリレーショナルデータベースだけでなく，リレーショナルデータベースとは全く異なるデータモデルを採用した，NoSQL と呼ばれるデータベース群が実用化されている昨今の状況を鑑みて，様々な方式のデータベースによる比較検討を行なっている．

現時点におけるプロダクトモデルの共有は，BIM/CIM が比較的進んでいる英国やフィンランドであっても，ファイルベースで実施されている場合がほとんどであるため，DBMS を用いたデータマネジメント技術に関する問題はまだ顕在化していない．本研究は，今後，必ず顕在化するはずである，データマネジメント技術に関する問題を明らかにし，さらに対策を講じたものである．これらの本研究の中心となる議論は，主に第 4 章，第 5 章，第 6 章でなされており，各章で得られた成果と結論を総括して以下に示す．

7.1.1 各章における検討結果のまとめ

(1) 各種データベースによるプロダクトモデル管理手法の開発

第 4 章では，IFC プロダクトモデルを以下に示す複数のデータモデルに変換する手法の開発に成功し，さらに，いくつかの DBMS 製品にプロダクトモデルを格納し，データ取得速度に関する実験を実施した．

- 関係モデル（リレーショナルデータベース）
- 半構造データ（XML データベース，ドキュメント指向データベース）
- プロパティグラフモデル（グラフデータベース）
- キー・バリューモデル（キー・バリューストア, カラム指向データベース）

表 7.1　プロダクトモデル管理手法の検討結果

データベース方式	データモデル	データ変換	データ取得性能
リレーショナル	関係モデル	可能だが困難	階層が深いデータで性能が低下
ドキュメント指向	半構造データ	可能	格納データ量に制限あり
グラフ型	プロパティグラフモデル	可能	良好
キー・バリューストア	キー・バリューモデル	非現実的	（未調査）

その結果（表 7.1）から，データ取得性能に関して，グラフデータベースが最も優れていると結論付けた．

(2)　グラフデータベースによるプロダクトモデル部分抽出手法の開発

第 5 章では，IFC プロダクトモデルを格納したグラフ DBMS における，部分抽出手法に関する検討を行なった．その結果，第 4 章で開発した変換手法によるデータには，部分抽出に必要な情報が不足していることが明らかになり，モデル変換手法の変更が必要になったが，最終的には，関係性を示すインスタンスをエッジで表現することで，より簡潔に IFC プロダクトモデルを表現可能になり，かつ，非常に簡潔なクエリで部分抽出が可能になった．

しかしながら，このモデル変換手法の変更により，継承情報を各ノードに保持することになり，データが冗長化することになってしまった．本研究の範囲では問題は発生していないが，DBMS のデータ容量を圧迫するのは間違いないため，今後は冗長化しない別の手法も検討すべきと思われる．

(3)　グラフ DBMS を用いた BIM/CIM 情報共有システムの提案

第 6 章では，BIM/CIM 情報共有システムを構築することを想定し，COCOMO 法を用いて開発コストを算出し，グラフ DBMS を用いたシステムの方が，開発コスト，及びメンテナンスコストが低くなることを明らかにした．

7.1.2　プロダクトモデルとデータベースの適合性に関する結論

以上に示したように，性能，データ変換，開発コストと様々な側面から検討した結果，グラフデータベースはすべてにおいて良好な結果が出ており，BIM/CIM において，あらゆる情報が集約されるプロダクトモデルと適合性が高いデータベース方式であると結論付けられる．これは，単にデータ構造の類似性に依るだけではなく，IFC スキーマの設計上の特徴である関係性の表現が，プロパティグラフモデルとうまく合致したからだと考えている．

IFC スキーマは今後も開発が続けられ，IFC5, IFC6, ... と進化していくと予想される．その過程で，IFC 拡張スキーマがさらに増えるのか，それとも集約されて 1 つになるのか，予見することはできないが，IFC スキーマの設計方針が大きく変化しない限り，IFC スキーマの新バージョン，派生バージョンが公開されたとしても，本論で示したグラフデータベースを用いたデータマネジメ

ント手法は，そのまま利用できるはずである．

しかしながら，本論における結論は，グラフデータベースの利点を明らかにしたものの，問題点を浮び上がらせるようなものではなかった．理論上，グラフデータベースの性能は，探索対象のグラフの大きさに比例して低下するはずであるが，そのような結果を確認することはできなかった．BIM/CIM 情報共有システムとしての実用化を想定するのであれば，取り扱い可能なデータのサイズについても検討されるべきであり，それが本研究の反省点である．

7.2 今後の課題と展望

7.2.1 データマネジメント技術に関する課題

本論では，グラフデータベースのデータモデルであるプロパティグラフモデルに対する部分抽出手法を研究した．理由は，BIM/CIM 情報共有システムで巨大なモデルを共有するには，部分抽出処理が必要不可欠だからである．しかし，部分抽出処理だけで十分であるはずはなく，その他に，

- 部分更新
- 差分検出
- 変更履歴管理

といった処理も必要だろう．差分検出が実現できれば，複数作業者による協調作業が大きく効率化すると考えられるし，変更履歴の管理技術は，維持管理フェーズにおいて非常に有用な機能を実現することになるだろう．

よって，データマネジメント技術に関しては，プロパティグラフモデルで表現したプロダクトモデルにおいて，これらを実現することが今後の課題となる．また，本論はデータベースとデータマネジメント技術に着目したため，これらデータの可視化について論じることはできなかったが，システムとして実用化するためには可視化技術も必要であり，これも今後の研究課題となり得る．

7.2.2 BIM/CIM 情報共有システムに関する課題

第 6 章で，BIM/CIM 情報共有システムのシステム構成として，リレーショナル DBMS とグラフ DBMS を併用するハイブリッド構成のシステムを提案した．本論でも論じたように，あらゆる構造のデータを効率良く扱える万能なデータベースは存在しない．よって，様々な種類のデータが必要な BIM/CIM 情報共有システムにおいて，取り扱うデータに合わせて DBMS を使い分けることは不自然ではなく，むしろ必然と言えるのではないだろうか．

ここ 20 年ほど，リレーショナル型以外に実用的（性能が高く，堅牢）な DBMS 製品が存在しなかったため，リレーショナル DBMS のみを用いてシステム化することが半ば常識となっていたように思われるが，その状況は，NoSQL データベースの台頭で変わりつつあると感じている．将来，実用的な NoSQL DBMS 製品が増えれば，1 つのシステム内で異なる方式の DBMS を併用するシステム構築手法が一般的になるかもしれない．

よって，今後はハイブリッド構成のシステムにおけるデータマネジメント技術，具体的には，異なる DBMS 間でのデータ授受に関する研究も必要になると考えている．

7.2.3　今後の展望

現在，日本では Part21 ファイルを用いたプロダクトモデルの運用に関する標準化が開始されたばかりで，まだファイルベースのやり取りが中心であるが，近い将来，BIM/CIM 情報共有システムの利用が一般化し，ネットワーク上でプロダクトモデルを共有し，維持管理までも含めたあらゆる情報が集約される日が訪れると信じている．そのような BIM/CIM の理想を実現すべく，研究を継続していく所存である．

謝辞

本論文は，大阪大学大学院工学研究科，矢吹信喜教授のご指導のもとに実施した研究をとりまとめたものです．私は，1997 年 3 月に新潟大学工学部を卒業してから，2014 年 4 月に大阪大学大学院に入学するまでの 17 年間，大学や学会活動にはまったく縁がなく，ほぼ何の実績もない状態から学位取得への取り組みを始めることになりました．そのため，矢吹先生からは，論文の内容云々以前に，文章の書き方からご指導いただいたような状態で，本当に世話のかかる学生だったと思いますが，懇切丁寧で，かつ熱意あるご指導を賜わり，感謝の念に堪えません．また，副査の澤木昌典教授，福田知弘准教授には，暖かく，かつ丁寧なコメントを多数いただきまして，誠にありがとうございました．この場を借りまして，深く御礼申し上げます．

この 3 年半の間は，私が籍を置く，川田テクノシステム株式会社における業務と並行しての研究活動でしたが，テクニカルイノベーションセンターの浦辺裕二部長，工藤克士次長（現在はエンタープライズソリューションセンター部長）には，私の研究活動に便宜を図っていただき，本当に助かりました．また，同部署の渡辺省嗣次長，豊田純教課長には，私の業務をフォローしていただくことが多々ありました．申し訳なかったと思うと同時に，心から感謝しています．

私は，1997 年に川田テクノシステム株式会社に入社しましたが，そこで，始めに興味を持ったことはオブジェクト指向設計/プログラミングでした．同時に，UML やデザインパターンの習得にも没頭しました．その後，XML データベースを利用したファイリングシステムの設計とプロトタイプの開発業務を通じて，当時の最新データベース技術に触れることができました．また，2006 年頃からは工事情報共有システムの開発とシステム運用に携わり，現在に至りますが，この業務を通して，リレーショナルデータベースやウェブシステムにも深く関わることになりました．本文中でも再三に渡って述べていますが，IFC プロダクトモデルはオブジェクト指向技術に基づいて設計されており，IFC プロダクトモデルとデータベースをテーマにした本論文は，3 年半に渡る研究の成果と言うだけでなく，私のこれまでの業務の集大成でもあると言えるかもしれません．よって，これまでに関わった，社外，社内を問わず，すべての方々のおかげで完成した論文であると思っています．ここに，すべての方々の名前を挙げることはできず，誠に申し訳ありませんが，本当にありがたく思っております．

また，学位取得のチャンスを与えてくださったばかりか，御自身が学位を取得した経験に基づいた有益なアドバイスもいただき，ときには励ましてくださった，川田テクノシステムの山野長弘社長に深く感謝の意を表します．今後は，研究を通じて得られた知識や技術を活用して，後輩指導や製品開発をすることで恩返しに代えさせていただきたい所存です．

最後になりますが，論文執筆中は休日に作業をすることも多く，一緒に出掛ける回数も減りましたが，不平も言わずに黙って私を支えてくれた妻の孝枝に感謝し，結びといたします．

2017 年 7 月

四月朔日（わたぬき） 勉

付録 A

プログラムコード

A.1　csv → obj 変換プログラム

ソースコード A.1　main.java

```
package net.qb13.mk3d;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.PrintStream;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.List;

import org.apache.commons.cli.BasicParser;
import org.apache.commons.cli.CommandLine;
import org.apache.commons.cli.HelpFormatter;
import org.apache.commons.cli.Options;
import org.apache.commons.cli.ParseException;
import org.apache.commons.lang3.StringUtils;


/**
 * 3Dのフォーマットを出力するユーティリティ
 * <p>
 * CSVをobj形式に変換する
 * </p>
 * <ol>
 * <li>2016/06/17: 新規作成
 * </ol>
 * @author nuki
 */
public class Main {

    public static void main(String args[]) throws Exception{
        // コマンドライン引数の仕様
        Options options = new Options();

        options.addOption("h", "help", false, "print usage.");

        try {
            // コマンドラインの解析
```

```
            CommandLine cl = new BasicParser().parse(options, args);

            if(cl.hasOption("help") || cl.getArgs().length == 0) {
                printUsage(options);
                return;
            }

            csv2obj(Paths.get(cl.getArgs()[0]), System.out);

        }
        catch(ParseException e) {
            printUsage(options);
        }
    }

    /**
     * CSVをobj形式に変換する
     * @param csvFile CSVファイルパス
     * @param os 出力先ストリーム
     * @throws IOException CSVファイルのオープンに失敗した
     */
    static void csv2obj(Path csvFile, PrintStream os) throws
IOException {
        // ファイルを読み込む
        try(BufferedReader reader = Files.newBufferedReader(csvFile)) {
            String line, faceId = null;
            List<Long> vertex = new ArrayList<Long>();
            long vn = 0;
            while((line = reader.readLine()) != null) {
                String[] columns = line.split(",");

                // 1面出力する
                if(!columns[0].equals(faceId) && !vertex.isEmpty())
                    printFace(os, vertex);

                faceId = columns[0];

                // 1点出力する
                os.printf("v %.20f %.20f %.20f\n",
                        Double.valueOf(StringUtils.remove(columns[1],
'"')),
                        Double.valueOf(StringUtils.remove(columns[2],
'"')),
                        Double.valueOf(StringUtils.remove(columns[3],
'"')));

                vertex.add(++vn);
            }

            // 最後の面を出力して完了
            printFace(os, vertex);
        }
    }

    protected static void printFace(PrintStream os, List<Long> vertex)
{
        os.print("f ");
        vertex.forEach(v -> os.print(v + " "));
        os.println();

        // データに不正があったらコメント行にワーニングを出力する
        if(hasErrors(vertex)) {
            os.println("# warning");
        }
```

```
        vertex.clear();;
    }

    /**
     * 面になっていない等の不整合を発見する
     * @param vertex
     * @return
     */
    static private boolean hasErrors(List<Long> vertex) {



        return false;
    }

    /**
     * 利用法の表示
     * @param options コマンドラインの仕様
     */
    static void printUsage(Options options) {
        HelpFormatter formatter =new HelpFormatter();

        System.out.println("mk3D (C) 2016-17 Tsutomu Watanuki");
        System.out.println("");
        formatter.printHelp("mk3d [options] <CSVファイル>", options);
        System.out.println("");
        System.out.println("例: mk3d < sample.csv > sample.obj");
    }
}
```

A.2 Data Access Object (Java 言語)

ソースコード A.2 Dao.java

```
package model.dao;

import model.entity.IfcGloballyUniqueId;

/**
 * Data Access Objectのインターフェイス
 */
public interface Dao<T> {

    /**
     * エンティティを追加
     * @param data 追加するエンティティ
     * @return 追加したエンティティのglobalId
     * @throws 追加に失敗した
     */
    public IfcGloballyUniqueId create(T data) throws Exception;

    /**
     * エンティティの取得
     * @param grobalId 取得するエンティティのgrobalId
     * @return 取得したエンティティ, 取得に失敗した場合はnull
     * @throws 取得に失敗した
     */
    public T get(IfcGloballyUniqueId globalId) throws Exception;
```

```
    /**
     * エンティティの更新
     * @param data 更新するエンティティ
     * @throws Exception 存在しないglobalIdが指定された
     */
    public void update(T data) throws Exception;

    /**
     * エンティティの削除
     * @param grobalId 削除するエンティティのgrobalId
     * @throws 削除に失敗した
     */
    public void delete(IfcGloballyUniqueId globalId) throws Exception;

    /**
     * 接続先のDBMS
     * @return 接続先
     */
    public default Object getConnection() {
        // 接続先のDBMSに合わせてimplementsする側で実装する
        return null;
    };
}
```

ソースコード A.3 IfcReinforcingBarDao.java（リレーショナル DBMS 用 DAO）

```
package model.dao;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

import model.entity.IfcGloballyUniqueId;
import model.entity.IfcReinforcingBar;
import model.entity.IfcReinforcingBar.IfcReinforcingBarSurfaceEnum;
import model.entity.IfcReinforcingBar.IfcReinforcingBarTypeEnum;

/**
 * IfcReinforcingBar エンティティのDAO
 *
 * <p>以下のテーブルにアクセスする</p>
 * <ol>
 * <li> ifc_root
 * <li> ifc_object
 * <li> ifc_product
 * <li> ifc_element
 * <li> ifc_reinforcing_element
 * <li> ifc_reinforce_bar
 * </ol>
 */
public class IfcReinforcingBarDao implements Dao<IfcReinforcingBar> {

    /**
     * IfcReinforcingBarを追加
     * @param data 追加するIfcReinforcingBar
     * @return 追加したデータのglobalId
     * @throws 追加に失敗した
     */
    public IfcGloballyUniqueId create(IfcReinforcingBar data) throws
Exception {

        Connection con = (Connection)getConnection();
```

```
        IfcGloballyUniqueId ifcReinforcingBarId = null;
        PreparedStatement stmt = null;

        // IfcRoot
        stmt = con.prepareStatement(CREATE_IFC_ROT_SQL);
        stmt.setString(1, data.getGlobalId().toString());
        stmt.setString(2, data.getOwnerHistory().toString());
        stmt.setString(3, data.getName());
        stmt.setString(4, data.getDescription());

        stmt.executeUpdate();
        stmt.close();

        // IfcObject
        stmt = con.prepareStatement(CREATE_IFC_OBJECT_SQL);
        stmt.setString(1, data.getGlobalId().toString());
        stmt.setString(2, data.getObjectType());

        stmt.executeUpdate();
        stmt.close();

        // IfcProduct
        stmt = con.prepareStatement(CREATE_IFC_PRODUCT_SQL);
        stmt.setString(1, data.getGlobalId().toString());
        stmt.setString(2, data.getObjectPlacement().toString());
        stmt.setString(3, data.getRepresentation().toString());

        stmt.executeUpdate();
        stmt.close();

        // IfcElement
        stmt = con.prepareStatement(CREATE_IFC_ELEMENT_SQL);
        stmt.setString(1, data.getGlobalId().toString());
        stmt.setString(2, data.getTag());

        stmt.executeUpdate();
        stmt.close();

        // IfcReinforcingElement
        stmt = con.prepareStatement(CREATE_IFC_REINFORCING_ELEMENT_SQL
);
        stmt.setString(1, data.getGlobalId().toString());
        stmt.setString(2, data.getSteelGrade());

        stmt.executeUpdate();
        stmt.close();

        // IfcReinforceBar
        stmt = con.prepareStatement(CREATE_IFC_REINFORCE_BAR_SQL);
        stmt.setString(1, data.getGlobalId().toString());
        stmt.setDouble(2, data.getNomicalDiameter());
        stmt.setDouble(3, data.getCrossSectionArea());
        stmt.setDouble(4, data.getBarLength());
        stmt.setString(5, data.getPredefinedType().name());
        stmt.setString(6, data.getBarSurface().name());

        stmt.executeUpdate();
        stmt.close();

        ifcReinforcingBarId = data.getGlobalId();

        return ifcReinforcingBarId;
    }

    /**
```

```
     * IfcReinforcingBarの取得
     * @param grobalId 取得するIfcReinforcingBarのgrobalId
     * @return 取得したIfcReinforcingBar, 取得に失敗した場合はnull
     * @throws 取得に失敗した
     */
    public IfcReinforcingBar get(IfcGloballyUniqueId grobalId) throws
Exception{
        Connection con = (Connection)getConnection();
        IfcReinforcingBar ifcReinforcingBar = null;
        PreparedStatement stmt = null;
        ResultSet result = null;

        stmt = con.prepareStatement(SELECT_SQL);
        stmt.setString(1, grobalId.toString());
        result = stmt.executeQuery();
        if (result.next()){
            ifcReinforcingBar = new IfcReinforcingBar();
            ifcReinforcingBar.setGlobalId(new IfcGloballyUniqueId(
result.getString(1)));
            ifcReinforcingBar.setOwnerHistory(new IfcGloballyUniqueId(
result.getString(2)));
            ifcReinforcingBar.setName(result.getString(3));
            ifcReinforcingBar.setDescription(result.getString(4));
            ifcReinforcingBar.setObjectType(result.getString(5));
            ifcReinforcingBar.setObjectPlacement(new
IfcGloballyUniqueId(result.getString(6)));
            ifcReinforcingBar.setRepresentation(new IfcGloballyUniqueId
(result.getString(7)));
            ifcReinforcingBar.setTag(result.getString(8));
            ifcReinforcingBar.setSteelGrade(result.getString(9));
            ifcReinforcingBar.setNomicalDiameter(result.getDouble(10));
            ifcReinforcingBar.setCrossSectionArea(result.getDouble
(11));
            ifcReinforcingBar.setBarLength(result.getDouble(12));
            ifcReinforcingBar.setPredefinedType(
IfcReinforcingBarTypeEnum.valueOf(result.getString(13)));
            ifcReinforcingBar.setBarSurface(
IfcReinforcingBarSurfaceEnum.valueOf(result.getString(14)));
        }

        return ifcReinforcingBar;
    }

    /**
     * IfcReinforcingBarの更新
     * @param data 更新するIfcReinforcingBar
     * @throws Exception 存在しないglobalIdが指定された
     */
    public void update(IfcReinforcingBar data) throws Exception{
        Connection con = (Connection)getConnection();
        PreparedStatement stmt = null;

        // IfcRoot
        stmt = con.prepareStatement(UPDATE_IFC_ROOT_SQL);
        stmt.setString(2, data.getOwnerHistory().toString());
        stmt.setString(3, data.getName());
        stmt.setString(4, data.getDescription());
        stmt.setString(5, data.getGlobalId().toString());
        int rowCount = stmt.executeUpdate();
        stmt.close();
        if(rowCount == 0){
            throw new Exception("Update Error: global_id: " + data.
getGlobalId());
        }
```

```
        // IfcObject
        stmt = con.prepareStatement(UPDATE_IFC_OBJECT_SQL);
        stmt.setString(2, data.getObjectType());
        stmt.setString(3, data.getGlobalId().toString());
        rowCount = stmt.executeUpdate();
        stmt.close();
        if(rowCount == 0){
            throw new Exception("Update Error: global_id: " + data.
getGlobalId());
        }

        // IfcProduct
        stmt = con.prepareStatement(UPDATE_IFC_PRODUCT_SQL);
        stmt.setString(2, data.getObjectPlacement().toString());
        stmt.setString(3, data.getRepresentation().toString());
        stmt.setString(4, data.getGlobalId().toString());
        rowCount = stmt.executeUpdate();
        stmt.close();
        if(rowCount == 0){
            throw new Exception("Update Error: global_id: " + data.
getGlobalId());
        }

        // IfcElement
        stmt = con.prepareStatement(UPDATE_IFC_ELEMENT_SQL);
        stmt.setString(2, data.getTag());
        stmt.setString(3, data.getGlobalId().toString());
        rowCount = stmt.executeUpdate();
        stmt.close();
        if(rowCount == 0){
            throw new Exception("Update Error: global_id: " + data.
getGlobalId());
        }

        // IfcReinforcingElement
        stmt = con.prepareStatement(UPDATE_IFC_REINFORCING_ELEMENT_SQL
);
        stmt.setString(2, data.getSteelGrade());
        stmt.setString(3, data.getGlobalId().toString());
        rowCount = stmt.executeUpdate();
        stmt.close();
        if(rowCount == 0){
            throw new Exception("Update Error: global_id: " + data.
getGlobalId());
        }

        // IfcReinforcingBar
        stmt = con.prepareStatement(UPDATE_IFC_REINFORCE_BAR_SQL);
        stmt.setDouble(2,  data.getNomicalDiameter());
        stmt.setDouble(3, data.getCrossSectionArea());
        stmt.setDouble(4, data.getBarLength());
        stmt.setString(5, data.getPredefinedType().name());
        stmt.setString(6, data.getBarSurface().name());
        stmt.setString(7, data.getGlobalId().toString());
        rowCount = stmt.executeUpdate();
        stmt.close();
        if(rowCount == 0){
            throw new Exception("Update Error: global_id: " + data.
getGlobalId());
        }
    }

    /**
```

```
     * IfcReinforcingBarの削除
     * @param grobalId 削除するIfcReinforcingBarのgrobalId
     * @throws 削除に失敗した
     */
    public void delete(IfcGloballyUniqueId grobalId) throws Exception{

        Connection con = (Connection)getConnection();
        PreparedStatement stmt = null;

        stmt = con.prepareStatement(DELETE_IFC_ROOT_SQL);
        stmt.setString(1, grobalId.toString());
        stmt.executeUpdate();
        stmt.close();

        stmt = con.prepareStatement(DELETE_IFC_OBJECT_SQL);
        stmt.setString(1, grobalId.toString());
        stmt.executeUpdate();
        stmt.close();

        stmt = con.prepareStatement(DELETE_IFC_PRODUCT_SQL);
        stmt.setString(1, grobalId.toString());
        stmt.executeUpdate();
        stmt.close();

        stmt = con.prepareStatement(DELETE_IFC_ELEMENT_SQL);
        stmt.setString(1, grobalId.toString());
        stmt.executeUpdate();
        stmt.close();

        stmt = con.prepareStatement(DELETE_IFC_REINFORCING_ELEMENT_SQL
);
        stmt.setString(1, grobalId.toString());
        stmt.executeUpdate();
        stmt.close();

        stmt = con.prepareStatement(DELETE_IFC_REINFORCE_BAR_SQL);
        stmt.setString(1, grobalId.toString());
        stmt.executeUpdate();
        stmt.close();
    }

     ////////////////////////////////////////////////
    // クエリ

    /**
     * 作成クエリ
     */
    private static String CREATE_IFC_ROT_SQL = "insert into ifc_root"
            + "( grobal_id, owner_history, name, description) "
            + "values "
            + "( ?, ?, ?, ? )";
    private static String CREATE_IFC_OBJECT_SQL = "insert into
ifc_object"
            + "( grobal_id, object_type) "
            + "values "
            + "( ?, ? )";
    private static String CREATE_IFC_PRODUCT_SQL = "insert into
ifc_product"
            + "( grobal_id, object_placement, representation) "
            + "values "
            + "( ?, ?, ? )";
    private static String CREATE_IFC_ELEMENT_SQL = "insert into
ifc_element"
            + "( grobal_id, tag) "
```

```
            + "values "
            + "( ?, ? )";
    private static String CREATE_IFC_REINFORCING_ELEMENT_SQL = "insert
into ifc_reinforcing_element"
            + "( grobal_id, steel_grade) "
            + "values "
            + "( ?, ? )";
    private static String CREATE_IFC_REINFORCE_BAR_SQL = "insert into
ifc_reinforce_bars"
            + "( grobal_id, nomical_diameter, cross_section_area,
bar_length, predefined_type, bar_surface) "
            + "values "
            + "( ?, ?, ?, ?, ?, ? )";

    /**
     * 取得クエリ
     */
    private static String SELECT_SQL = "select "
            + "ifc_root.grobal_id, ifc_root.owner_history, ifc_root.
name, ifc_root.description, "
            + "ifc_object.object_type, "
            + "ifc_product.object_placement, ifc_product.representation
, "
            + "ifc_element.tag, "
            + "ifc_reinforcing_element.steel_grade, "
            + "ifc_reinforcing_bar.nomical_diameter,
ifc_reinforcing_bar.cross_section_area, ifc_reinforcing_bar.bar_length,
 ifc_reinforcing_bar.predefined_type, ifc_reinforcing_bar.bar_surface "
            + "from ifc_root, ifc_object, ifc_product, ifc_element,
ifc_reinforcing_element, ifc_reinforce_bar"
            + "where ifc_root.grobal_id = ? and ifc_object.grobal_id =
? and ifc_product.grobal_id = ? and ifc_element.grobal_id = ? and
ifc_reinforcing_element.grobal_id = ? and ifc_reinforcing_bar = ?";

    /**
     * 更新クエリ
     */
    private static String UPDATE_IFC_ROOT_SQL = "update ifc_root "
            + "set owner_history = ?, name = ?, description = ? "
            + "where grobal_id = ?";
    private static String UPDATE_IFC_OBJECT_SQL = "update ifc_object "
            + "set object_type = ? "
            + "where grobal_id = ?";
    private static String UPDATE_IFC_PRODUCT_SQL = "update ifc_product
"
            + "set object_placement = ?, representation = ? "
            + "where grobal_id = ?";
    private static String UPDATE_IFC_ELEMENT_SQL = "update ifc_element
"
            + "set tag = ? "
            + "where grobal_id = ?";
    private static String UPDATE_IFC_REINFORCING_ELEMENT_SQL = "update
ifc_reinforcing_element "
            + "set steel_grade = ? "
            + "where grobal_id = ?";
    private static String UPDATE_IFC_REINFORCE_BAR_SQL = "update
ifc_reinforce_bar "
            + "set nomical_diameter = ?, cross_section_area = ?,
bar_length = ?, predefined_type = ?, bar_surface = ? "
            + "where grobal_id = ?";

    /**
     * 削除クエリ
     */
```

```
    private static String DELETE_IFC_ROOT_SQL = "delete from ifc_root
where global_id = ? ";
    private static String DELETE_IFC_OBJECT_SQL = "delete from
ifc_object where global_id = ? ";
    private static String DELETE_IFC_PRODUCT_SQL = "delete from
ifc_product where global_id = ? ";
    private static String DELETE_IFC_ELEMENT_SQL = "delete from
ifc_element where global_id = ? ";
    private static String DELETE_IFC_REINFORCING_ELEMENT_SQL = "delete
from ifc_reinforcing_element where global_id = ? ";
    private static String DELETE_IFC_REINFORCE_BAR_SQL = "delete from
ifc_reinforce_bar where global_id = ? ";
}
```

ソースコード A.4 IfcGraphDao.java（グラフ DBMS 用 DAO）

```
package model.dao;

import java.beans.Introspector;
import java.beans.PropertyDescriptor;

import model.entity.IfcGloballyUniqueId;
import model.entity.IfcReinforcingBar.IfcReinforcingBarSurfaceEnum;
import model.entity.IfcReinforcingBar.IfcReinforcingBarTypeEnum;
import model.entity.IfcRoot;

import org.apache.commons.beanutils.BeanUtils;
import org.apache.commons.beanutils.ConvertUtils;
import org.apache.commons.beanutils.Converter;
import org.neo4j.driver.v1.AuthTokens;
import org.neo4j.driver.v1.Driver;
import org.neo4j.driver.v1.GraphDatabase;
import org.neo4j.driver.v1.Session;
import org.neo4j.driver.v1.StatementResult;
import org.neo4j.driver.v1.exceptions.NoSuchRecordException;
import org.neo4j.driver.v1.types.Node;

/**
 * グラフDBMS(Neo4j)用のData Access Objectクラス.
 * @param <T> 対象オブジェクトの型
 */
public class IfcGraphDao<T extends IfcRoot> implements Dao<T> {

    /**
     * コンストラクタ
     * @param clazz 対象クラス名
     */
    public IfcGraphDao(Class<T> clazz) {
        _class = clazz;
    }

    private Class<T> _class;

    /**
     * オブジェクト追加
     * @param data 追加するオブジェクト
     * @retern 追加したオブジェクトのID
     */
    @Override
    public IfcGloballyUniqueId create(T data) throws Exception {
        // クエリ構築
        StringBuffer query = new StringBuffer();
        query.append("CREATE (n:");
```

```
        query.append(_class.getSimpleName());
        query.append(" {");

        for(PropertyDescriptor prop : Introspector.getBeanInfo(_class).
getPropertyDescriptors()) {
            if(prop.getWriteMethod() == null)
                continue;

            Object value = getPropertyValue(data, prop.getName());
            if(value == null)
                continue;

            query.append(prop.getName());
            query.append(":");
            query.append(value);
            query.append(",");
        }

        query.deleteCharAt(query.length() - 1);
        query.append("})");

        // クエリ実行
        try(Session session = getConnection().session()) {
            session.run(query.toString());
        }

        return data.getGlobalId();
    }

    private Object getPropertyValue(Object bean, String propName)
throws Exception {
        Object value = BeanUtils.getProperty(bean, propName);
        if(value == null || value instanceof Number)
            return value;

        if(value instanceof IfcRoot) {
            // エンティティ
            return getPropertyValue(value, "globalId");
        }
        else {
            return "\"" + value + "\"";
        }
    }

    /**
     * オブジェクト取得
     * @param id 取得対象のID
     * @return 取得したオブジェクト（存在しなかった場合null）
     */
    @Override
    public T get(IfcGloballyUniqueId id) throws Exception {
        try {
            String query = String.format("MATCH(n) WHERE n.globalId=\"%
s\" RETURN n;", id.get());

            StatementResult result = getConnection().session().run(
query);
            Node node = result.single().get(0).asNode();

            ConvertUtils.register(new GloballyUniqueIdConverter(),
IfcGloballyUniqueId.class);
            ConvertUtils.register(new EnumConverter<
IfcActionRequestTypeEnum>(IfcActionRequestTypeEnum.class),
IfcActionRequestTypeEnum.class);
```

```
            ConvertUtils.register(new EnumConverter<
IfcActionSourceTypeEnum>(IfcActionSourceTypeEnum.class),
IfcActionSourceTypeEnum.class);
            ConvertUtils.register(new EnumConverter<IfcActionTypeEnum>(
IfcActionTypeEnum.class), IfcActionTypeEnum.class);
            ConvertUtils.register(new EnumConverter<IfcActuatorTypeEnum
>(IfcActuatorTypeEnum.class), IfcActuatorTypeEnum.class);
            ConvertUtils.register(new EnumConverter<IfcAddressTypeEnum
>(IfcAddressTypeEnum.class), IfcAddressTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcAirTerminalBoxTypeEnum>(IfcAirTerminalBoxTypeEnum.class),
IfcAirTerminalBoxTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcAirTerminalTypeEnum>(IfcAirTerminalTypeEnum.class),
IfcAirTerminalTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcAirToAirHeatRecoveryTypeEnum>(IfcAirToAirHeatRecoveryTypeEnum.class
), IfcAirToAirHeatRecoveryTypeEnum.class);
            ConvertUtils.register(new EnumConverter<IfcAlarmTypeEnum>(
IfcAlarmTypeEnum.class), IfcAlarmTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcAnalysisModelTypeEnum>(IfcAnalysisModelTypeEnum.class),
IfcAnalysisModelTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcAnalysisTheoryTypeEnum>(IfcAnalysisTheoryTypeEnum.class),
IfcAnalysisTheoryTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcArithmeticOperatorEnum>(IfcArithmeticOperatorEnum.class),
IfcArithmeticOperatorEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcAssemblyPlaceEnum>(IfcAssemblyPlaceEnum.class), IfcAssemblyPlaceEnum
.class);
            ConvertUtils.register(new EnumConverter<
IfcAudioVisualApplianceTypeEnum>(IfcAudioVisualApplianceTypeEnum.class
), IfcAudioVisualApplianceTypeEnum.class);
            ConvertUtils.register(new EnumConverter<IfcBeamTypeEnum>(
IfcBeamTypeEnum.class), IfcBeamTypeEnum.class);
            ConvertUtils.register(new EnumConverter<IfcBenchmarkEnum>(
IfcBenchmarkEnum.class), IfcBenchmarkEnum.class);
            ConvertUtils.register(new EnumConverter<IfcBoilerTypeEnum>(
IfcBoilerTypeEnum.class), IfcBoilerTypeEnum.class);
            ConvertUtils.register(new EnumConverter<IfcBooleanOperator
>(IfcBooleanOperator.class), IfcBooleanOperator.class);
            ConvertUtils.register(new EnumConverter<IfcBSplineCurveForm
>(IfcBSplineCurveForm.class), IfcBSplineCurveForm.class);
            ConvertUtils.register(new EnumConverter<
IfcBSplineSurfaceForm>(IfcBSplineSurfaceForm.class),
IfcBSplineSurfaceForm.class);
            ConvertUtils.register(new EnumConverter<
IfcBuildingElementPartTypeEnum>(IfcBuildingElementPartTypeEnum.class),
IfcBuildingElementPartTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcBuildingElementProxyTypeEnum>(IfcBuildingElementProxyTypeEnum.class
), IfcBuildingElementProxyTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcBuildingSystemTypeEnum>(IfcBuildingSystemTypeEnum.class),
IfcBuildingSystemTypeEnum.class);
            ConvertUtils.register(new EnumConverter<IfcBurnerTypeEnum>(
IfcBurnerTypeEnum.class), IfcBurnerTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcCableCarrierFittingTypeEnum>(IfcCableCarrierFittingTypeEnum.class),
IfcCableCarrierFittingTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcCableCarrierSegmentTypeEnum>(IfcCableCarrierSegmentTypeEnum.class),
IfcCableCarrierSegmentTypeEnum.class);
```

```
            ConvertUtils.register(new EnumConverter<
IfcCableFittingTypeEnum>(IfcCableFittingTypeEnum.class),
IfcCableFittingTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcCableSegmentTypeEnum>(IfcCableSegmentTypeEnum.class),
IfcCableSegmentTypeEnum.class);
            ConvertUtils.register(new EnumConverter<IfcChangeActionEnum
>(IfcChangeActionEnum.class), IfcChangeActionEnum.class);
            ConvertUtils.register(new EnumConverter<IfcChillerTypeEnum
>(IfcChillerTypeEnum.class), IfcChillerTypeEnum.class);
            ConvertUtils.register(new EnumConverter<IfcChimneyTypeEnum
>(IfcChimneyTypeEnum.class), IfcChimneyTypeEnum.class);
            ConvertUtils.register(new EnumConverter<IfcCoilTypeEnum>(
IfcCoilTypeEnum.class), IfcCoilTypeEnum.class);
            ConvertUtils.register(new EnumConverter<IfcColumnTypeEnum>(
IfcColumnTypeEnum.class), IfcColumnTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcCommunicationsApplianceTypeEnum>(IfcCommunicationsApplianceTypeEnum.
class), IfcCommunicationsApplianceTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcComplexPropertyTemplateTypeEnum>(IfcComplexPropertyTemplateTypeEnum.
class), IfcComplexPropertyTemplateTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcCompressorTypeEnum>(IfcCompressorTypeEnum.class),
IfcCompressorTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcCondenserTypeEnum>(IfcCondenserTypeEnum.class), IfcCondenserTypeEnum
.class);
            ConvertUtils.register(new EnumConverter<
IfcConnectionTypeEnum>(IfcConnectionTypeEnum.class),
IfcConnectionTypeEnum.class);
            ConvertUtils.register(new EnumConverter<IfcConstraintEnum>(
IfcConstraintEnum.class), IfcConstraintEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcConstructionEquipmentResourceTypeEnum>(
IfcConstructionEquipmentResourceTypeEnum.class),
IfcConstructionEquipmentResourceTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcConstructionMaterialResourceTypeEnum>(
IfcConstructionMaterialResourceTypeEnum.class),
IfcConstructionMaterialResourceTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcConstructionProductResourceTypeEnum>(
IfcConstructionProductResourceTypeEnum.class),
IfcConstructionProductResourceTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcControllerTypeEnum>(IfcControllerTypeEnum.class),
IfcControllerTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcCooledBeamTypeEnum>(IfcCooledBeamTypeEnum.class),
IfcCooledBeamTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcCoolingTowerTypeEnum>(IfcCoolingTowerTypeEnum.class),
IfcCoolingTowerTypeEnum.class);
            ConvertUtils.register(new EnumConverter<IfcCostItemTypeEnum
>(IfcCostItemTypeEnum.class), IfcCostItemTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcCostScheduleTypeEnum>(IfcCostScheduleTypeEnum.class),
IfcCostScheduleTypeEnum.class);
            ConvertUtils.register(new EnumConverter<IfcCoveringTypeEnum
>(IfcCoveringTypeEnum.class), IfcCoveringTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcCrewResourceTypeEnum>(IfcCrewResourceTypeEnum.class),
IfcCrewResourceTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
```

```
IfcCurtainWallTypeEnum>(IfcCurtainWallTypeEnum.class),
IfcCurtainWallTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcCurveInterpolationEnum>(IfcCurveInterpolationEnum.class),
IfcCurveInterpolationEnum.class);
            ConvertUtils.register(new EnumConverter<IfcDamperTypeEnum>(
IfcDamperTypeEnum.class), IfcDamperTypeEnum.class);
            ConvertUtils.register(new EnumConverter<IfcDataOriginEnum>(
IfcDataOriginEnum.class), IfcDataOriginEnum.class);
            ConvertUtils.register(new EnumConverter<IfcDerivedUnitEnum
>(IfcDerivedUnitEnum.class), IfcDerivedUnitEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcDirectionSenseEnum>(IfcDirectionSenseEnum.class),
IfcDirectionSenseEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcDiscreteAccessoryTypeEnum>(IfcDiscreteAccessoryTypeEnum.class),
IfcDiscreteAccessoryTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcDistributionChamberElementTypeEnum>(
IfcDistributionChamberElementTypeEnum.class),
IfcDistributionChamberElementTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcDistributionPortTypeEnum>(IfcDistributionPortTypeEnum.class),
IfcDistributionPortTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcDistributionSystemEnum>(IfcDistributionSystemEnum.class),
IfcDistributionSystemEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcDocumentConfidentialityEnum>(IfcDocumentConfidentialityEnum.class),
IfcDocumentConfidentialityEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcDocumentStatusEnum>(IfcDocumentStatusEnum.class),
IfcDocumentStatusEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcDoorPanelOperationEnum>(IfcDoorPanelOperationEnum.class),
IfcDoorPanelOperationEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcDoorPanelPositionEnum>(IfcDoorPanelPositionEnum.class),
IfcDoorPanelPositionEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcDoorStyleConstructionEnum>(IfcDoorStyleConstructionEnum.class),
IfcDoorStyleConstructionEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcDoorStyleOperationEnum>(IfcDoorStyleOperationEnum.class),
IfcDoorStyleOperationEnum.class);
            ConvertUtils.register(new EnumConverter<IfcDoorTypeEnum>(
IfcDoorTypeEnum.class), IfcDoorTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcDoorTypeOperationEnum>(IfcDoorTypeOperationEnum.class),
IfcDoorTypeOperationEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcDuctFittingTypeEnum>(IfcDuctFittingTypeEnum.class),
IfcDuctFittingTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcDuctSegmentTypeEnum>(IfcDuctSegmentTypeEnum.class),
IfcDuctSegmentTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcDuctSilencerTypeEnum>(IfcDuctSilencerTypeEnum.class),
IfcDuctSilencerTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcElectricApplianceTypeEnum>(IfcElectricApplianceTypeEnum.class),
IfcElectricApplianceTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcElectricDistributionBoardTypeEnum>(
IfcElectricDistributionBoardTypeEnum.class),
```

```
IfcElectricDistributionBoardTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcElectricFlowStorageDeviceTypeEnum>(
IfcElectricFlowStorageDeviceTypeEnum.class),
IfcElectricFlowStorageDeviceTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcElectricGeneratorTypeEnum>(IfcElectricGeneratorTypeEnum.class),
IfcElectricGeneratorTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcElectricMotorTypeEnum>(IfcElectricMotorTypeEnum.class),
IfcElectricMotorTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcElectricTimeControlTypeEnum>(IfcElectricTimeControlTypeEnum.class),
IfcElectricTimeControlTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcElementAssemblyTypeEnum>(IfcElementAssemblyTypeEnum.class),
IfcElementAssemblyTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcElementCompositionEnum>(IfcElementCompositionEnum.class),
IfcElementCompositionEnum.class);
            ConvertUtils.register(new EnumConverter<IfcEngineTypeEnum>(
IfcEngineTypeEnum.class), IfcEngineTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcEvaporativeCoolerTypeEnum>(IfcEvaporativeCoolerTypeEnum.class),
IfcEvaporativeCoolerTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcEvaporatorTypeEnum>(IfcEvaporatorTypeEnum.class),
IfcEvaporatorTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcEventTriggerTypeEnum>(IfcEventTriggerTypeEnum.class),
IfcEventTriggerTypeEnum.class);
            ConvertUtils.register(new EnumConverter<IfcEventTypeEnum>(
IfcEventTypeEnum.class), IfcEventTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcExternalSpatialElementTypeEnum>(IfcExternalSpatialElementTypeEnum.
class), IfcExternalSpatialElementTypeEnum.class);
            ConvertUtils.register(new EnumConverter<IfcFanTypeEnum>(
IfcFanTypeEnum.class), IfcFanTypeEnum.class);
            ConvertUtils.register(new EnumConverter<IfcFastenerTypeEnum
>(IfcFastenerTypeEnum.class), IfcFastenerTypeEnum.class);
            ConvertUtils.register(new EnumConverter<IfcFilterTypeEnum>(
IfcFilterTypeEnum.class), IfcFilterTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcFireSuppressionTerminalTypeEnum>(IfcFireSuppressionTerminalTypeEnum.
class), IfcFireSuppressionTerminalTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcFlowDirectionEnum>(IfcFlowDirectionEnum.class), IfcFlowDirectionEnum
.class);
            ConvertUtils.register(new EnumConverter<
IfcFlowInstrumentTypeEnum>(IfcFlowInstrumentTypeEnum.class),
IfcFlowInstrumentTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcFlowMeterTypeEnum>(IfcFlowMeterTypeEnum.class), IfcFlowMeterTypeEnum
.class);
            ConvertUtils.register(new EnumConverter<IfcFootingTypeEnum
>(IfcFootingTypeEnum.class), IfcFootingTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcFurnitureTypeEnum>(IfcFurnitureTypeEnum.class), IfcFurnitureTypeEnum
.class);
            ConvertUtils.register(new EnumConverter<
IfcGeographicElementTypeEnum>(IfcGeographicElementTypeEnum.class),
IfcGeographicElementTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcGeometricProjectionEnum>(IfcGeometricProjectionEnum.class),
IfcGeometricProjectionEnum.class);
```

```
            ConvertUtils.register(new EnumConverter<
IfcGlobalOrLocalEnum>(IfcGlobalOrLocalEnum.class), IfcGlobalOrLocalEnum
.class);
            ConvertUtils.register(new EnumConverter<IfcGridTypeEnum>(
IfcGridTypeEnum.class), IfcGridTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcHeatExchangerTypeEnum>(IfcHeatExchangerTypeEnum.class),
IfcHeatExchangerTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcHumidifierTypeEnum>(IfcHumidifierTypeEnum.class),
IfcHumidifierTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcInterceptorTypeEnum>(IfcInterceptorTypeEnum.class),
IfcInterceptorTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcInternalOrExternalEnum>(IfcInternalOrExternalEnum.class),
IfcInternalOrExternalEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcInventoryTypeEnum>(IfcInventoryTypeEnum.class), IfcInventoryTypeEnum
.class);
            ConvertUtils.register(new EnumConverter<
IfcJunctionBoxTypeEnum>(IfcJunctionBoxTypeEnum.class),
IfcJunctionBoxTypeEnum.class);
            ConvertUtils.register(new EnumConverter<IfcKnotType>(
IfcKnotType.class), IfcKnotType.class);
            ConvertUtils.register(new EnumConverter<
IfcLaborResourceTypeEnum>(IfcLaborResourceTypeEnum.class),
IfcLaborResourceTypeEnum.class);
            ConvertUtils.register(new EnumConverter<IfcLampTypeEnum>(
IfcLampTypeEnum.class), IfcLampTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcLayerSetDirectionEnum>(IfcLayerSetDirectionEnum.class),
IfcLayerSetDirectionEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcLightDistributionCurveEnum>(IfcLightDistributionCurveEnum.class),
IfcLightDistributionCurveEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcLightEmissionSourceEnum>(IfcLightEmissionSourceEnum.class),
IfcLightEmissionSourceEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcLightFixtureTypeEnum>(IfcLightFixtureTypeEnum.class),
IfcLightFixtureTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcLoadGroupTypeEnum>(IfcLoadGroupTypeEnum.class), IfcLoadGroupTypeEnum
.class);
            ConvertUtils.register(new EnumConverter<
IfcLogicalOperatorEnum>(IfcLogicalOperatorEnum.class),
IfcLogicalOperatorEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcMechanicalFastenerTypeEnum>(IfcMechanicalFastenerTypeEnum.class),
IfcMechanicalFastenerTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcMedicalDeviceTypeEnum>(IfcMedicalDeviceTypeEnum.class),
IfcMedicalDeviceTypeEnum.class);
            ConvertUtils.register(new EnumConverter<IfcMemberTypeEnum>(
IfcMemberTypeEnum.class), IfcMemberTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcMotorConnectionTypeEnum>(IfcMotorConnectionTypeEnum.class),
IfcMotorConnectionTypeEnum.class);
            ConvertUtils.register(new EnumConverter<IfcNullStyle>(
IfcNullStyle.class), IfcNullStyle.class);
            ConvertUtils.register(new EnumConverter<IfcObjectiveEnum>(
IfcObjectiveEnum.class), IfcObjectiveEnum.class);
            ConvertUtils.register(new EnumConverter<IfcObjectTypeEnum>(
IfcObjectTypeEnum.class), IfcObjectTypeEnum.class);
```

```
            ConvertUtils.register(new EnumConverter<IfcOccupantTypeEnum
>(IfcOccupantTypeEnum.class), IfcOccupantTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcOpeningElementTypeEnum>(IfcOpeningElementTypeEnum.class),
IfcOpeningElementTypeEnum.class);
            ConvertUtils.register(new EnumConverter<IfcOutletTypeEnum>(
IfcOutletTypeEnum.class), IfcOutletTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcPerformanceHistoryTypeEnum>(IfcPerformanceHistoryTypeEnum.class),
IfcPerformanceHistoryTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcPermeableCoveringOperationEnum>(IfcPermeableCoveringOperationEnum.
class), IfcPermeableCoveringOperationEnum.class);
            ConvertUtils.register(new EnumConverter<IfcPermitTypeEnum>(
IfcPermitTypeEnum.class), IfcPermitTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcPhysicalOrVirtualEnum>(IfcPhysicalOrVirtualEnum.class),
IfcPhysicalOrVirtualEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcPileConstructionEnum>(IfcPileConstructionEnum.class),
IfcPileConstructionEnum.class);
            ConvertUtils.register(new EnumConverter<IfcPileTypeEnum>(
IfcPileTypeEnum.class), IfcPileTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcPipeFittingTypeEnum>(IfcPipeFittingTypeEnum.class),
IfcPipeFittingTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcPipeSegmentTypeEnum>(IfcPipeSegmentTypeEnum.class),
IfcPipeSegmentTypeEnum.class);
            ConvertUtils.register(new EnumConverter<IfcPlateTypeEnum>(
IfcPlateTypeEnum.class), IfcPlateTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcProcedureTypeEnum>(IfcProcedureTypeEnum.class), IfcProcedureTypeEnum
.class);
            ConvertUtils.register(new EnumConverter<IfcProfileTypeEnum
>(IfcProfileTypeEnum.class), IfcProfileTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcProjectedOrTrueLengthEnum>(IfcProjectedOrTrueLengthEnum.class),
IfcProjectedOrTrueLengthEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcProjectionElementTypeEnum>(IfcProjectionElementTypeEnum.class),
IfcProjectionElementTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcProjectOrderTypeEnum>(IfcProjectOrderTypeEnum.class),
IfcProjectOrderTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcPropertySetTemplateTypeEnum>(IfcPropertySetTemplateTypeEnum.class),
IfcPropertySetTemplateTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcProtectiveDeviceTrippingUnitTypeEnum>(
IfcProtectiveDeviceTrippingUnitTypeEnum.class),
IfcProtectiveDeviceTrippingUnitTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcProtectiveDeviceTypeEnum>(IfcProtectiveDeviceTypeEnum.class),
IfcProtectiveDeviceTypeEnum.class);
            ConvertUtils.register(new EnumConverter<IfcPumpTypeEnum>(
IfcPumpTypeEnum.class), IfcPumpTypeEnum.class);
            ConvertUtils.register(new EnumConverter<IfcRailingTypeEnum
>(IfcRailingTypeEnum.class), IfcRailingTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcRampFlightTypeEnum>(IfcRampFlightTypeEnum.class),
IfcRampFlightTypeEnum.class);
            ConvertUtils.register(new EnumConverter<IfcRampTypeEnum>(
IfcRampTypeEnum.class), IfcRampTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
```

```
IfcRecurrenceTypeEnum>(IfcRecurrenceTypeEnum.class),
IfcRecurrenceTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcReflectanceMethodEnum>(IfcReflectanceMethodEnum.class),
IfcReflectanceMethodEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcReinforcingBarRoleEnum>(IfcReinforcingBarRoleEnum.class),
IfcReinforcingBarRoleEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcReinforcingBarSurfaceEnum>(IfcReinforcingBarSurfaceEnum.class),
IfcReinforcingBarSurfaceEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcReinforcingBarTypeEnum>(IfcReinforcingBarTypeEnum.class),
IfcReinforcingBarTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcReinforcingMeshTypeEnum>(IfcReinforcingMeshTypeEnum.class),
IfcReinforcingMeshTypeEnum.class);
            ConvertUtils.register(new EnumConverter<IfcRoleEnum>(
IfcRoleEnum.class), IfcRoleEnum.class);
            ConvertUtils.register(new EnumConverter<IfcRoofTypeEnum>(
IfcRoofTypeEnum.class), IfcRoofTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcSanitaryTerminalTypeEnum>(IfcSanitaryTerminalTypeEnum.class),
IfcSanitaryTerminalTypeEnum.class);
            ConvertUtils.register(new EnumConverter<IfcSectionTypeEnum
>(IfcSectionTypeEnum.class), IfcSectionTypeEnum.class);
            ConvertUtils.register(new EnumConverter<IfcSensorTypeEnum>(
IfcSensorTypeEnum.class), IfcSensorTypeEnum.class);
            ConvertUtils.register(new EnumConverter<IfcSequenceEnum>(
IfcSequenceEnum.class), IfcSequenceEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcShadingDeviceTypeEnum>(IfcShadingDeviceTypeEnum.class),
IfcShadingDeviceTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcSimplePropertyTemplateTypeEnum>(IfcSimplePropertyTemplateTypeEnum.
class), IfcSimplePropertyTemplateTypeEnum.class);
            ConvertUtils.register(new EnumConverter<IfcSIPrefix>(
IfcSIPrefix.class), IfcSIPrefix.class);
            ConvertUtils.register(new EnumConverter<IfcSIUnitName>(
IfcSIUnitName.class), IfcSIUnitName.class);
            ConvertUtils.register(new EnumConverter<IfcSlabTypeEnum>(
IfcSlabTypeEnum.class), IfcSlabTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcSolarDeviceTypeEnum>(IfcSolarDeviceTypeEnum.class),
IfcSolarDeviceTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcSpaceHeaterTypeEnum>(IfcSpaceHeaterTypeEnum.class),
IfcSpaceHeaterTypeEnum.class);
            ConvertUtils.register(new EnumConverter<IfcSpaceTypeEnum>(
IfcSpaceTypeEnum.class), IfcSpaceTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcSpatialZoneTypeEnum>(IfcSpatialZoneTypeEnum.class),
IfcSpatialZoneTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcStackTerminalTypeEnum>(IfcStackTerminalTypeEnum.class),
IfcStackTerminalTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcStairFlightTypeEnum>(IfcStairFlightTypeEnum.class),
IfcStairFlightTypeEnum.class);
            ConvertUtils.register(new EnumConverter<IfcStairTypeEnum>(
IfcStairTypeEnum.class), IfcStairTypeEnum.class);
            ConvertUtils.register(new EnumConverter<IfcStateEnum>(
IfcStateEnum.class), IfcStateEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcStructuralCurveActivityTypeEnum>(IfcStructuralCurveActivityTypeEnum.
```

```
class), IfcStructuralCurveActivityTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcStructuralCurveMemberTypeEnum>(IfcStructuralCurveMemberTypeEnum.
class), IfcStructuralCurveMemberTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcStructuralSurfaceActivityTypeEnum>(
IfcStructuralSurfaceActivityTypeEnum.class),
IfcStructuralSurfaceActivityTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcStructuralSurfaceMemberTypeEnum>(IfcStructuralSurfaceMemberTypeEnum.
class), IfcStructuralSurfaceMemberTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcSubContractResourceTypeEnum>(IfcSubContractResourceTypeEnum.class),
IfcSubContractResourceTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcSurfaceFeatureTypeEnum>(IfcSurfaceFeatureTypeEnum.class),
IfcSurfaceFeatureTypeEnum.class);
            ConvertUtils.register(new EnumConverter<IfcSurfaceSide>(
IfcSurfaceSide.class), IfcSurfaceSide.class);
            ConvertUtils.register(new EnumConverter<
IfcSwitchingDeviceTypeEnum>(IfcSwitchingDeviceTypeEnum.class),
IfcSwitchingDeviceTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcSystemFurnitureElementTypeEnum>(IfcSystemFurnitureElementTypeEnum.
class), IfcSystemFurnitureElementTypeEnum.class);
            ConvertUtils.register(new EnumConverter<IfcTankTypeEnum>(
IfcTankTypeEnum.class), IfcTankTypeEnum.class);
            ConvertUtils.register(new EnumConverter<IfcTaskDurationEnum
>(IfcTaskDurationEnum.class), IfcTaskDurationEnum.class);
            ConvertUtils.register(new EnumConverter<IfcTaskTypeEnum>(
IfcTaskTypeEnum.class), IfcTaskTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcTendonAnchorTypeEnum>(IfcTendonAnchorTypeEnum.class),
IfcTendonAnchorTypeEnum.class);
            ConvertUtils.register(new EnumConverter<IfcTendonTypeEnum>(
IfcTendonTypeEnum.class), IfcTendonTypeEnum.class);
            ConvertUtils.register(new EnumConverter<IfcTextPath>(
IfcTextPath.class), IfcTextPath.class);
            ConvertUtils.register(new EnumConverter<
IfcTimeSeriesDataTypeEnum>(IfcTimeSeriesDataTypeEnum.class),
IfcTimeSeriesDataTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcTransformerTypeEnum>(IfcTransformerTypeEnum.class),
IfcTransformerTypeEnum.class);
            ConvertUtils.register(new EnumConverter<IfcTransitionCode>(
IfcTransitionCode.class), IfcTransitionCode.class);
            ConvertUtils.register(new EnumConverter<
IfcTransportElementTypeEnum>(IfcTransportElementTypeEnum.class),
IfcTransportElementTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcTrimmingPreference>(IfcTrimmingPreference.class),
IfcTrimmingPreference.class);
            ConvertUtils.register(new EnumConverter<
IfcTubeBundleTypeEnum>(IfcTubeBundleTypeEnum.class),
IfcTubeBundleTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcUnitaryControlElementTypeEnum>(IfcUnitaryControlElementTypeEnum.
class), IfcUnitaryControlElementTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcUnitaryEquipmentTypeEnum>(IfcUnitaryEquipmentTypeEnum.class),
IfcUnitaryEquipmentTypeEnum.class);
            ConvertUtils.register(new EnumConverter<IfcUnitEnum>(
IfcUnitEnum.class), IfcUnitEnum.class);
            ConvertUtils.register(new EnumConverter<IfcValveTypeEnum>(
IfcValveTypeEnum.class), IfcValveTypeEnum.class);
```

```
            ConvertUtils.register(new EnumConverter<
IfcVibrationIsolatorTypeEnum>(IfcVibrationIsolatorTypeEnum.class),
IfcVibrationIsolatorTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcVoidingFeatureTypeEnum>(IfcVoidingFeatureTypeEnum.class),
IfcVoidingFeatureTypeEnum.class);
            ConvertUtils.register(new EnumConverter<IfcWallTypeEnum>(
IfcWallTypeEnum.class), IfcWallTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcWasteTerminalTypeEnum>(IfcWasteTerminalTypeEnum.class),
IfcWasteTerminalTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcWindowPanelOperationEnum>(IfcWindowPanelOperationEnum.class),
IfcWindowPanelOperationEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcWindowPanelPositionEnum>(IfcWindowPanelPositionEnum.class),
IfcWindowPanelPositionEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcWindowStyleConstructionEnum>(IfcWindowStyleConstructionEnum.class),
IfcWindowStyleConstructionEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcWindowStyleOperationEnum>(IfcWindowStyleOperationEnum.class),
IfcWindowStyleOperationEnum.class);
            ConvertUtils.register(new EnumConverter<IfcWindowTypeEnum>(
IfcWindowTypeEnum.class), IfcWindowTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcWindowTypePartitioningEnum>(IfcWindowTypePartitioningEnum.class),
IfcWindowTypePartitioningEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcWorkCalendarTypeEnum>(IfcWorkCalendarTypeEnum.class),
IfcWorkCalendarTypeEnum.class);
            ConvertUtils.register(new EnumConverter<IfcWorkPlanTypeEnum
>(IfcWorkPlanTypeEnum.class), IfcWorkPlanTypeEnum.class);
            ConvertUtils.register(new EnumConverter<
IfcWorkScheduleTypeEnum>(IfcWorkScheduleTypeEnum.class),
IfcWorkScheduleTypeEnum.class);

            // オブジェクト構築
            T instance = _class.newInstance();
            for(String key : node.keys())
                BeanUtils.setProperty(instance, key, node.get(key).
asString());

            return instance;
        }
        catch(NoSuchRecordException ex) {
            return null;
        }
    }

    /**
     * オブジェクト更新
     * @param data 更新対象オブジェクト
     */
    @Override
    public void update(T data) throws Exception {
        // クエリ構築
        StringBuffer query = new StringBuffer();
        query.append("MATCH(n) WHERE n.globalId=\"");
        query.append(data.getGlobalId());
        query.append("\" SET ");

        for(PropertyDescriptor prop : Introspector.getBeanInfo(_class).
getPropertyDescriptors()) {
```

```
            if(prop.getWriteMethod() == null)
                continue;

            Object value = getPropertyValue(data, prop.getName());
            if(value == null)
                continue;

            query.append("n.");
            query.append(prop.getName());
            query.append("=");
            query.append(value);
            query.append(",");
        }
        query.deleteCharAt(query.length() - 1);

        // クエリ実行
        try(Session session = getConnection().session()) {
            session.run(query.toString());
        }
    }

    /**
     * オブジェクト削除
     * @param id 削除対象オブジェクトのID
     */
    @Override
    public void delete(IfcGloballyUniqueId id) throws Exception {
        // クエリ構築
        String query = String.format("MATCH(n) WHERE n.globalId=\"%s\"
DELETE n;", id.get());

        // クエリ実行
        try(Session session = getConnection().session()) {
            session.run(query);
        }
    }

     ////////////////////////////////////////////////////////////
    // コンバータ

    private class GloballyUniqueIdConverter implements Converter {
        @Override @SuppressWarnings({"unchecked", "hiding"})
        public <T> T convert(Class<T> type, Object value) {
            return (T)new IfcGloballyUniqueId((String)value);
        }
    }

    private class EnumConverter<E extends Enum<E>> implements Converter
 {

        public EnumConverter(Class<E> enumClass) {
            _enumClass = enumClass;
        }

        private Class<E> _enumClass;

        @SuppressWarnings({"unchecked", "hiding"})
        public <T> T convert(Class<T> type, Object value) {
            return (T)Enum.valueOf(_enumClass, (String)value);
        }
    }
}
```

ソースコード A.5 IfcRoot.java

```
package model.entity;

/**
 * IfcRoot エンティティ
 */
public class IfcRoot {

    private IfcGloballyUniqueId _globalId = new IfcGloballyUniqueId();
    private IfcGloballyUniqueId _ownerHistory;
    private String _name;
    private String _description;

     ////////////////////////////////////////////////
    // アクセサメソッド

    public IfcGloballyUniqueId getGlobalId() {
        return _globalId;
    }

    public void setGlobalId(IfcGloballyUniqueId grobalId) {
        this._globalId = grobalId;
    }

    public IfcGloballyUniqueId getOwnerHistory() {
        return _ownerHistory;
    }

    public void setOwnerHistory(IfcGloballyUniqueId ownerHistory) {
        this._ownerHistory = ownerHistory;
    }

    public String getName() {
        return _name;
    }

    public void setName(String name) {
        this._name = name;
    }

    public String getDescription() {
        return _description;
    }

    public void setDescription(String description) {
        this._description = description;
    }
}
```

ソースコード A.6 IfcObjectDefinition.java

```
package model.entity;

/**
 * IfcObjectDefinition エンティティ
 */
public class IfcObjectDefinition extends IfcRoot{

}
```

ソースコード A.7　IfcObject.java

```
package model.entity;

/**
 * IfcObject エンティティ
 */
public class IfcObject extends IfcObjectDefinition{

    private String _objectType;

     ////////////////////////////////////////////////
    // アクセサメソッド

    public String getObjectType() {
        return _objectType;
    }

    public void setObjectType(String objectType) {
        this._objectType = objectType;
    }
}
```

ソースコード A.8　IfcProduct.java

```
package model.entity;

/**
 * IfcProduct エンティティ
 */
public class IfcProduct extends IfcObject{
    private IfcGloballyUniqueId _objectPlacement;
    private IfcGloballyUniqueId _representation;

     ////////////////////////////////////////////////
    // アクセサメソッド

    public IfcGloballyUniqueId getObjectPlacement() {
        return _objectPlacement;
    }

    public void setObjectPlacement(IfcGloballyUniqueId objectPlacement)
 {
        this._objectPlacement = objectPlacement;
    }

    public IfcGloballyUniqueId getRepresentation() {
        return _representation;
    }

    public void setRepresentation(IfcGloballyUniqueId representation) {
        this._representation = representation;
    }
}
```

ソースコード A.9　IfcElement.java

```
package model.entity;

/**
 * IfcElement エンティティ
 */
public class IfcElement extends IfcProduct{
```

```
    private String _tag;

     ////////////////////////////////////////////////
    // アクセサメソッド

    public String getTag() {
        return _tag;
    }
    public void setTag(String tag) {
        this._tag = tag;
    }
}
```

ソースコード A.10 IfcElementComponent.java

```
package model.entity;

/**
 * IfcElementComponent エンティティ
 */
public class IfcElementComponent extends IfcElement{

}
```

ソースコード A.11 IfcReinforcingElement.java

```
package model.entity;

/**
 * IfcReinforcingElement エンティティ
 */
public class IfcReinforcingElement extends IfcElementComponent{

    private String _steelGrade;

     ////////////////////////////////////////////////
    // アクセサメソッド

    public String getSteelGrade() {
        return _steelGrade;
    }

    public void setSteelGrade(String steelGrade) {
        this._steelGrade = steelGrade;
    }
}
```

ソースコード A.12 IfcReinforcingBar.java

```
package model.entity;

/**
 * IfcReinforcingBar エンティティ
 */
public class IfcReinforcingBar extends IfcReinforcingElement{

    public enum IfcReinforcingBarTypeEnum{
        ANCHORING,
        EDGE,
        LIGATURE,
        MAIN,
```

```
        PUNCHING,
        RING,
        SHEAR,
        STUD,
        USERDEFINED,
        NOTDEFINED;
    }

    public enum IfcReinforcingBarSurfaceEnum{
        PLAIN,
        TEXTURED;
    }

     ////////////////////////////////////////////////
    // フィールド

    private Double _nomicalDiameter;
    private Double _crossSectionArea;
    private Double _barLength;
    private IfcReinforcingBarTypeEnum _predefinedType;
    private IfcReinforcingBarSurfaceEnum _barSurface;

     ////////////////////////////////////////////////
    // アクセサメソッド

    public Double getNomicalDiameter() {
        return _nomicalDiameter;
    }

    public void setNomicalDiameter(Double _nomicalDiameter) {
        this._nomicalDiameter = _nomicalDiameter;
    }

    public Double getCrossSectionArea() {
        return _crossSectionArea;
    }

    public void setCrossSectionArea(Double crossSectionArea) {
        this._crossSectionArea = crossSectionArea;
    }

    public Double getBarLength() {
        return _barLength;
    }

    public void setBarLength(Double barLength) {
        this._barLength = barLength;
    }

    public IfcReinforcingBarTypeEnum getPredefinedType() {
        return _predefinedType;
    }

    public void setPredefinedType(IfcReinforcingBarTypeEnum
predefinedType) {
        this._predefinedType = predefinedType;
    }

    public IfcReinforcingBarSurfaceEnum getBarSurface() {
        return _barSurface;
    }

    public void setBarSurface(IfcReinforcingBarSurfaceEnum barSurface)
{
```

```
        this._barSurface = barSurface;
    }
}
```

ソースコード A.13 IfcGloballyUniqueId.java

```
package model.entity;

import java.util.Base64;
import java.util.UUID;

public class IfcGloballyUniqueId {

    public IfcGloballyUniqueId() {
        _id = Base64.getEncoder().encodeToString(UUID.randomUUID().
toString().getBytes());
    }

    public IfcGloballyUniqueId(String id) {
        _id = id;
    }

    public String get() {
        return _id;
    }

    private String _id;

    @Override
    public String toString() {
        return _id;
    }

    @Override
    public boolean equals(Object obj) {
        if(!(obj instanceof IfcGloballyUniqueId))
            return false;

        return this.toString().equals(obj.toString());
    }
}
```

付録 B

関係スキーマ（テーブル定義）

ソースコード B.1　IfcBridge.sql

```
CREATE TABLE [dbo].[IfcBridge](
    [GlobalId] [nchar](32) NOT NULL,
    [OwnerHistory] [nchar](32) NULL,
    [Name] [nvarchar](max) NULL,
    [Description] [nvarchar](max) NULL,
    [StructureIndicator] [nvarchar](max) NULL,
    [CompositionType] [nvarchar](max) NULL,
    [StructureType] [nvarchar](max) NULL,
 CONSTRAINT [PK_IfcBridge] PRIMARY KEY CLUSTERED
(
    [GlobalId] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
```

ソースコード B.2　IfcBridgePart.sql

```
CREATE TABLE [dbo].[IfcBridgePart](
    [GlobalId] [nchar](32) NOT NULL,
    [OwnerHistory] [nchar](32) NULL,
    [Name] [nvarchar](max) NULL,
    [Description] [nvarchar](max) NULL,
    [StructureIndicator] [nvarchar](50) NULL,
    [CompositionType] [nvarchar](50) NULL,
    [StructureElementType] [nvarchar](50) NULL,
    [TechnoElementType] [nvarchar](50) NULL,
 CONSTRAINT [PK_IfcBridgePart] PRIMARY KEY CLUSTERED
(
    [GlobalId] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
```

ソースコード B.3　IfcBridgePrismaticElement.sql

```
CREATE TABLE [dbo].[IfcBridgePrismaticElement](
    [GlobalId] [nchar](32) NOT NULL,
    [Name] [nvarchar](max) NULL,
    [Description] [nvarchar](max) NULL,
    [Tag] [nvarchar](max) NULL,
    [PrismaticElementType] [nvarchar](50) NULL,
```

```
 CONSTRAINT [PK_IfcBridgePrismaticElement] PRIMARY KEY CLUSTERED
(
    [GlobalId] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]

GO

ALTER TABLE [dbo].[IfcBridgePrismaticElement] ADD  CONSTRAINT [
DF_IfcBridgePrismaticElement_GlobalId]  DEFAULT (replace(lower(newid
()),'-','')) FOR [GlobalId]
GO
```

ソースコード B.4 IfcBridgeSegment.sql

```
CREATE TABLE [dbo].[IfcBridgeSegment](
    [GlobalId] [nchar](32) NOT NULL,
    [Name] [nvarchar](max) NULL,
    [Description] [nvarchar](max) NULL,
    [SegmentType] [nvarchar](50) NULL,
 CONSTRAINT [PK_IfcBridgeSegment] PRIMARY KEY CLUSTERED
(
    [GlobalId] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
```

ソースコード B.5 IfcCartesianPoint.sql

```
CREATE TABLE [dbo].[IfcCartesianPoint](
    [GlobalId] [nchar](32) NOT NULL,
    [Coordinates] [nchar](32) NOT NULL,
 CONSTRAINT [PK_IfcCartesianPoint] PRIMARY KEY CLUSTERED
(
    [GlobalId] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
```

ソースコード B.6 IfcCompositeCurve.sql

```
CREATE TABLE [dbo].[IfcCompositeCurve](
    [GlobalId] [nchar](32) NOT NULL,
    [Segments] [nchar](32) NOT NULL,
 CONSTRAINT [PK_IfcCompositeCurve] PRIMARY KEY CLUSTERED
(
    [GlobalId] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
```

ソースコード B.7 IfcCompositeCurveSegment.sql

```
CREATE TABLE [dbo].[IfcCompositeCurveSegment](
    [GlobalId] [nchar](32) NOT NULL,
    [ParentCurve] [nchar](32) NOT NULL,
 CONSTRAINT [PK_IfcCompositeCurveSegment] PRIMARY KEY CLUSTERED
(
    [GlobalId] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
```

ソースコード B.8　IfcPolyline.sql

```
CREATE TABLE [dbo].[IfcPolyline](
    [GlobalId] [nchar](32) NULL,
    [Points] [nchar](32) NULL
) ON [PRIMARY]
```

ソースコード B.9　IfcProductDefinitionShape.sql

```
CREATE TABLE [dbo].[IfcProductDefinitionShape](
    [GlobalId] [nchar](32) NOT NULL,
    [Representations] [nchar](32) NOT NULL,
 CONSTRAINT [PK_IfcProductDefinitionShape] PRIMARY KEY CLUSTERED
(
    [GlobalId] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
```

ソースコード B.10　IfcReinforcingBar.sql

```
CREATE TABLE [dbo].[IfcReinforcingBar](
    [GlobalId] [nchar](32) NOT NULL,
    [Name] [nvarchar](max) NULL,
    [Description] [nvarchar](max) NULL,
    [Tag] [nvarchar](max) NULL,
    [SteelGrade] [nvarchar](max) NULL,
    [NominalDiameter] [float] NULL,
    [CrossSectionArea] [float] NULL,
    [BarLength] [float] NULL,
    [BarRole] [nvarchar](50) NULL,
    [BarSurface] [nvarchar](50) NULL,
    [Representation] [nchar](32) NOT NULL,
 CONSTRAINT [PK_IfcReinforcingBar] PRIMARY KEY CLUSTERED
(
    [GlobalId] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]

GO

ALTER TABLE [dbo].[IfcReinforcingBar] ADD  CONSTRAINT [
DF_IfcReinforcingBar_GlobalId]  DEFAULT (replace(lower(newid
()),'-','')) FOR [GlobalId]
GO
```

ソースコード B.11　IfcRelAggregates.sql

```
CREATE TABLE [dbo].[IfcRelAggregates](
    [GlobalId] [nchar](32) NOT NULL,
    [Name] [nvarchar](255) NULL,
    [Description] [nvarchar](255) NULL,
    [OwnerHistoryId] [int] NULL,
    [RelatingObject] [nchar](32) NOT NULL,
    [RelatedElements] [nchar](32) NOT NULL
) ON [PRIMARY]

GO

ALTER TABLE [dbo].[IfcRelAggregates]  WITH CHECK ADD  CONSTRAINT [
CK_IfcRelAggregates_GlobalId] CHECK  ((len([GlobalId])>=(22)))
GO
```

```
ALTER TABLE [dbo].[IfcRelAggregates] CHECK CONSTRAINT [
CK_IfcRelAggregates_GlobalId]
GO
```

ソースコード B.12 IfcRelContainedInSpatialStructure.sql

```
CREATE TABLE [dbo].[IfcRelContainedInSpatialStructure](
    [GlobalId] [nchar](32) NOT NULL,
    [Name] [nvarchar](255) NULL,
    [Description] [nvarchar](255) NULL,
    [OwnerHistoryId] [int] NULL,
    [RelatingStructure] [nchar](32) NOT NULL,
    [RelatedElements] [nchar](32) NULL,
 CONSTRAINT [PK_IfcRelContainedInSpatialStructure_1] PRIMARY KEY
CLUSTERED
(
    [GlobalId] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

ALTER TABLE [dbo].[IfcRelContainedInSpatialStructure]  WITH CHECK ADD
CONSTRAINT [CK_IfcRelContainedInSpatialStructure_GlobalId] CHECK  ((len
([GlobalId])>=(22)))
GO

ALTER TABLE [dbo].[IfcRelContainedInSpatialStructure] CHECK CONSTRAINT
[CK_IfcRelContainedInSpatialStructure_GlobalId]
GO
```

ソースコード B.13 IfcRelNests.sql

```
CREATE TABLE [dbo].[IfcRelNests](
    [GlobalId] [nchar](32) NOT NULL,
    [Name] [nchar](10) NULL,
    [Description] [nchar](10) NULL,
    [RelatingObject] [nchar](32) NOT NULL,
    [RelatedElements] [nchar](32) NOT NULL,
 CONSTRAINT [PK_IfcRelNests] PRIMARY KEY CLUSTERED
(
    [GlobalId] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

ALTER TABLE [dbo].[IfcRelNests] ADD  CONSTRAINT [
DF_IfcRelNests_GlobalId]  DEFAULT (replace(lower(newid()),'-','')) FOR
[GlobalId]
GO
```

ソースコード B.14 IfcShapeRepresentation.sql

```
CREATE TABLE [dbo].[IfcShapeRepresentation](
    [GlobalId] [nchar](32) NOT NULL,
    [RepresentationIdentifier] [nvarchar](255) NULL,
    [RepresentationType] [nvarchar](255) NULL,
    [ContextOfItemsId] [int] NULL,
```

```
    [Items] [nchar](32) NOT NULL,
 CONSTRAINT [PK_IfcShapeRepresentation] PRIMARY KEY CLUSTERED
(
    [GlobalId] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
```

ソースコード B.15　IfcSweptDiskSolid.sql

```
CREATE TABLE [dbo].[IfcSweptDiskSolid](
    [GlobalId] [nchar](32) NOT NULL,
    [Directrix] [nchar](32) NOT NULL,
 CONSTRAINT [PK_IfcSweptDiskSolid] PRIMARY KEY CLUSTERED
(
    [GlobalId] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
```

ソースコード B.16　Related.sql

```
CREATE TABLE [dbo].[Related](
    [GlobalId] [nchar](32) NOT NULL,
    [RelatedObject] [nchar](32) NOT NULL,
 CONSTRAINT [PK_Related] PRIMARY KEY CLUSTERED
(
    [GlobalId] ASC,
    [RelatedObject] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY
= OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
```