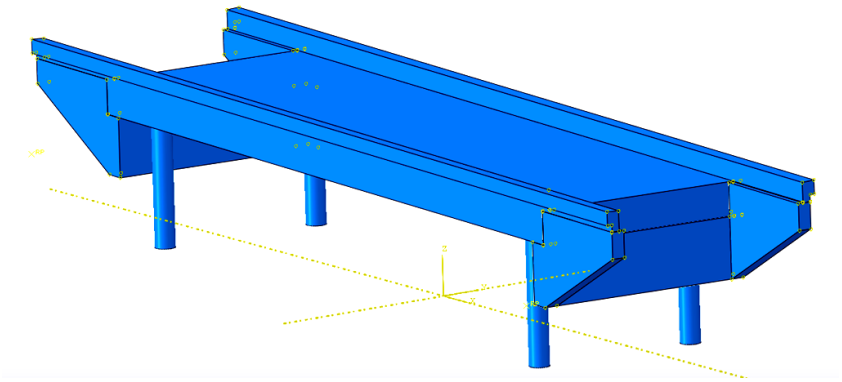
Effektivare projektering av brokonstruktioner med hjälp av BIM



Niklas Liberg

Avdelningen för Konstruktionsteknik
Lunds Tekniska Högskola
Lunds Universitet, 2012

Rapport TVBK - 5204

Avdelningen för Konstruktionsteknik
Lunds Tekniska Högskola
Box 118
221 00 LUND

Department of Structural Engineering
Lund Institute of Technology
Box 118
S-221 00 LUND
Sweden

Effektivare projektering av brokonstruktioner med hjälp av BIM

A more effective bridge design process utilizing BIM

Niklas Liberg

2012

Abstract

There‟s a notable trend within the building industry to move from traditional two-dimensional drawing tools towards three-dimensional design tools in order to make the design process more effective with a quicker and more rational way to produce blue prints and other documents. Parallel to this trend it‟s also become more common for structural engineers to use three dimensional tools for structural analysis. It is the same type of information which is used to create the blue print as well as the structural building model therefore making this an incentive for software developers in trying to come up with a way to automatically transfer building models between different types of applications as this is an obvious way to save expensive engineering time.

This Master‟s Project has been carried out with the purpose of analyzing the potential of a BIM design process for bridge structures with focus on hands on strategies for automatic transfer of building models between design tools and tools for structural analysis.

Compared to a two dimensional CAD building model, a parameterized three dimensional BIM model is composed in a entirely different and more sophisticated way as it rests on an object oriented data structure which opens up for a broad spectra of new technical applications such as automatic construction scheduling, billing of quantities and collision control. This new technique also opens up for an inter-disciplinary work in a shared building model making the design process more effective. However, discipline specific applications are generally not compatible with each other as software vendors usually employ proprietary file formats within their software environment. This report presents a method that enables automatic file transfer of building models via the neutral IFC file format.

The IFC building model is developed as a future standard for BIM models addressing every discipline in the field and is meant to be a living document throughout the construction"s entire life cycle from early sketching to demolishing. By programming and running an IFC interpreter, a tool that can read and transform the information from an IFC model into an internal representation, I have demonstrated a way to import a three dimensional geometry from a design tool called Revit Structure into a structural analysis tool called BRIGADE/Plus without the need for manual repair. This method suggests that BIM is a rational technique to go with as manual drafting takes time. However, the information transfer is associated with some disturbing technical issues which may not have any obvious solutions.

The IFC file format is in summary only as good as it can be implemented and utilized by its users. The IFC model might be a good alternative when it comes to selecting a standard for BIM models but it is very important that the IFC syntax is implemented in a correct way by software vendors and there might be a lot of technical issues hard to foresee as the model addresses so many different disciplines with vastly different information needs and requirements. Implementation of BIM in the building industry depends on flexibility, precision and application handiness.

There are at least three main criteria that have to be fulfilled in order for BIM to have its big breakthrough in the industry namely development and implementation of essential technique, efficient transfer of information and the establishment of a standard for shared BIM models.

Rapport TVBK - 5204
ISSN 0349-4969
ISRN LUTVGD/TVBK-12/5204(66)
Nyckelord: BIM, IFC, FEM, Objektorienterad modellering, Brokonstruktion

Examensarbete, Civ.Ing., Väg- och Vattenbyggnadsprogrammet 270hp

Handledare: Johan Kölfors och Anders Vennerstad, Scanscot Technolgy AB

Examinator: Miklós Mólnar, Universitetslektor, Lunds Universitet

Februari, 2012

Förord

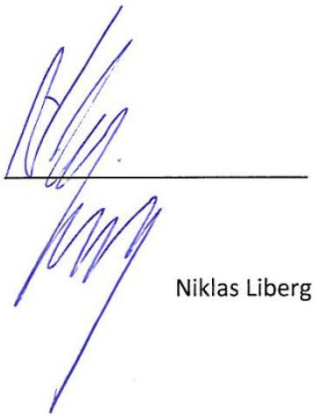
Rapporten är en dokumentation av mitt examensarbete på civilingenjörsutbildningen i Väg- och Vattenbyggnad på Lunds Tekniska Högskola som utfördes under våren 2011. Arbetet utformades av Scanscot Technology i syfte att ta fram ett underlag för framtida teknisk utveckling och bedrevs i huvudsak på företagets mycket trevliga kontor i Lund.

Jag vill tacka mina handledare på Scanscot Technology, Johan Kölfors och Anders Vennerstad, för en utomordentligt bra handledning som lett till mycket nyvunnen kunskap och ett gott resultat. Jag vill tacka alla vänner på Scanscot Technology för en jätterolig tid på kontoret som gav en mycket positiv bild av det stundande arbetslivet för en nyutexaminerad civilingenjör.

Jag vill också rikta uppmärksamhet mot mina tidigare klasskamrater Petter Hasselberg och John Argérus som bidragit med värdefull information och inspiration genom rapporten ”Dimensionering med BIM kopplat till FEM” som publicerats på avdelningen för byggnadsmekanik.

Examinator är Dr. Miklós Mólnar, universitetslektor vid avdelningen för konstruktionsteknik på LTH.

Malmö, februari 2012



Sammanfattning

Det blir allt vanligare att projektering inom byggbranschen utförs med hjälp av tredimensionella ritverktyg. Huvudsyftet är att effektivisera produktionen av ritningar och andra arbetshandlingar. Parallellt med denna utveckling blir det också allt vanligare att genomföra strukturanalys och dimensionering av broar med hjälp av tredimensionella beräkningsverktyg. Det är till stora delar samma information som används för att skapa den geometriska modellen i ritverktyget som sedan också används för att skapa beräkningsmodellen i strukturanalysverktyget. Den till stora delar manuella, dataöverföringen mellan olika mjukvaror är ett tidsödande arbete och därför skulle ett automatiserat informationsutbyte mellan dessa typer av verktyg rimligen leda till betydande effektiviseringsvinster.

Examensarbetet har utförts i syfte är att utreda och förklara hur BIM på ett konkret sätt kan effektivisera projekteringen av brokonstruktioner genom att automatisera överföringen av bromodeller mellan ritverktyg och verktyg för strukturanalys.

Jämfört traditionella, tvådimensionella CAD-modeller är en tredimensionell BIM-modell uppbyggd på ett fundamentalt annorlunda sätt med en objektorienterad datastruktur vilket öppnar upp för ett brett spektrum av nya tekniska tillämpningar exempelvis automatiserad mängdning, schemaläggning och kollisionskontroll. BIM är också en teknik som tillåter att flera discipliner arbetar samtidigt i en delad byggnadsmodell. Disciplinspecifika applikationer är dock generellt inte kompatibla med varandra men i den här rapporten presenteras en metod som möjliggör en automatiserad överföring av byggnadsmodeller via det neutrala IFC-formatet.

IFC-modellen är utformad för att kunna matcha alla branschens disciplinspecifika BIM-applikationer och verka som en standard för delade byggnadsmodeller genom hela byggprocessen från projektering till rivning. Genom att programmera en så kallad IFC-tolk, ett verktyg som tillåter att informationen i en IFC-modell importeras och omvandlas till en intern representation har jag bevisat hur man med hjälp av IFC kan överföra en tredimensionell geometri från projekteringsverktyget Revit Structure till strukturanalysverktyget BRIGADE/Plus utan manuell handpåläggning. Detta är en konkret effektivisering av projekteringsskedet för brokonstruktioner eftersom modelleringen är tidskrävande. Överföringen är dock behäftad med vissa tekniska problem som kanske inte har någon självklar lösning för det finns trots allt några viktiga skillnader mellan en konstruktionsritning och en beräkningsmodell.

IFC-formatet är sammanfattningsvis bara så bra som det implementeras och utnyttjas av sina användare. Om IFC-syntaxen kan implementeras på ett korrekt sätt är IFC-modellen möjligtvis ett bra alternativ som en BIM-standard men det finns sannolikt många tekniska problem i samband med utvecklingen av en disciplinöverskridande byggnadsmodell som först måste övervinnas. Anammandet av BIM-tekniken i byggbranschen är beroende av användarvänlighet, flexibilitet och precision.

Det finns åtminstone tre grundläggande förutsättningar som måste uppfyllas för att BIM skall slå igenom på bred front nämligen utveckling och implementering av nödvändig teknik, smidig och automatiserad filöverföring samt etablering av en BIM-standard för delade byggnadsmodeller.

Innehållsförteckning

1. Bakgrund 1
   1.1 Byggbranschen – en bransch i förändring? 1
   1.2 Om rapporten 2
2. Projekteringen av en brokonstruktion 5
3. Grundläggande begrepp 7
   3.1 BIM 7
   3.2 Objektorienterad modellering 9
   3.3 IFC 11
   3.4 Finita elementmetoden och verktyg för strukturanalys 12
4. Metodik 15
   4.1 Från konstruktionsritning till beräkningsmodell – eller tvärtom? 15
   4.2 Informationsdelning mellan olika applikationer 19
   4.3 IFC-modellen 20
   4.4 IFC-syntaxen 26
5. Implementering 37
   5.1 Utvecklingen av en prototyp för import av IFC-modeller 37
   5.2 Exporterande applikation – Autodesk Revit Structure 2011 39
   5.3 Importerande applikation - BRIGADE/Plus 3.1 40
   5.4 Prototypen 41
   5.5 Testning av prototypen 50
   5.6 Problemet med solidmodellering för strukturanalysen av brokonstruktioner 52
6 Slutsats och diskussion 53
   6.1 Implementering av nödvändig teknik 53
   6.2 Automatiserad modellöverföring 53
   6.3 Branschstandard för BIM-modeller 55
   6.4 BIM – fördelar och nackdelar 55
7 Referenser 57
8 Bilagor 58
   8.1 Brokonstruktionens geometri 58
   8.2 Lista över IFC-certifierade mjukvaror 59

1. Bakgrund

1.1 Byggbranschen – en bransch i förändring?

Det blir allt vanligare att projektering inom byggbranschen utförs med hjälp av tredimensionella ritverktyg. Huvudsyftet är att effektivisera produktionen av ritningar och andra arbetshandlingar. Parallellt med denna utveckling blir det också allt vanligare att genomföra strukturanalys och dimensionering av broar med hjälp av tredimensionella beräkningsverktyg. Det är till stora delar samma information som används för att skapa den geometriska modellen i ritverktyget som sedan också används för att skapa beräkningsmodellen i strukturanalysverktyget. Denna dubblerade modellering, dels i ritverktyget och dels i strukturanalysverktyget, är tidsödande och därför skulle ett automatiserat informationsutbyte mellan dessa typer av verktyg rimligen leda till betydande effektiviseringsvinster.

Tanken på att möjliggöra den här sortens disciplinöverskridande arbete och att övergå till att arbeta i en gemensam modell har funnits en lång tid i byggbranschen men utvecklingen har gått mycket långsamt jämfört andra tillverkningsbranscher. Inom exempelvis verkstadsindustrin, som jobbar med stora tillverkningsserier, finns uppenbara incitament för att effektivisera en given process i projekteringen och inom exempelvis bil- och flygindustrin är det automatiserade informationsutbytet mellan disciplinspecifika applikationer väl utvecklat sedan en lång tid tillbaka. Byggbranschen, i synnerhet brobyggnadsbranschen, däremot jobbar inte med serietillverkning utan ofta med helt ”unika” projekt, vilket bland annat innebär att de ekonomiska marginalerna som skulle kunna användas för att avsätta pengar till teknisk utveckling är betydligt mindre. Byggbranschen skylls också ofta för att vara en mycket konservativ bransch, inom vilken de olika disciplinerna är organiserade i en cementerad arbetsstruktur. Med detta sagt förefaller det ändå som om det i dagsläget finns en ökande lust och drivkraft inom byggbranschen för utvecklingen av ny teknik och ett utökat disciplinöverskridande samarbete.

BIM, eller *Building Information Modeling*, är ett relativt nytt begrepp inom byggbranschen som brukar lyftas fram i dessa sammanhang. BIM kan beskrivas som en teknik som utvecklas i syfte att effektivisera arbetet hela vägen genom byggnadens livscykel från ritbord till rivning. Visionen är att alla discipliner skall kunna dela med sig av sin och ta del av andras information i en och samma byggnadsmodell så att dubbelarbete och missförstånd undviks i större utsträckning. Med ett närmre samarbete mellan olika discipliner blir det också möjligt att arbeta på ett mera iterativt sätt då mindre tid behöver dediceras åt insamling och bearbetning av data och mer tid kan läggas på att pröva olika idéer. Detta borde leda till en högre kvalité och ekonomiska besparingar genom hela byggprocessen. I framtiden väntar möjligtvis en omfattande kulturell förändring av arbetet i byggbranschen.

Inledningsvis finns det en hel del tekniska problem som måste övervinnas. I den här rapporten presenteras en automatiserad metod som tillåter att en BIM-modell skapad i en BIM-applikation överförs och omvandlas till en beräkningsmodell i ett strukturanalysverktyg.

1.2 Om rapporten

På initiativ av Scanscot Technology i Lund har jag undersökt potentialen i BIM avseende effektivisering av arbetet att skapa beräkningsmodeller för brokonstruktioner.

Syfte

Rapportens syfte är att förklara hur BIM kan effektivisera projekteringen av brokonstruktioner och då särskilt i samband med informationsutbytet mellan ritverktyg och verktyg för strukturanalys. En problematisering av ämnet skulle kunna formuleras enligt frågan:

Hur utformas lämpligast en metodik för automatiserad import och omvandling av CAD-modeller till beräkningsmodeller i ett strukturanalysverktyg?

Målsättning

Målsättningen för examensarbetet var inledningsvis att redogöra för hur BIM lämpligast kan användas i samband med strukturanalysen av en brokonstruktion. Detta skulle konkretiseras av att programmera en prototyp för automatiserad import av BIM-modeller till Scanscots strukturanalysverktyg *BRIGADE/Plus.*

Målsättningen för rapporten är att läsaren skall bli introducerad till begreppet BIM samt få en något djupare teknisk förståelse för relaterat material exempelvis *objektorienterad modellering* och *IFC-formatet*. Rapporten är i första hand en dokumentation av mitt examensarbete men det är också min ambition att denna rapport skall vara en vägledning för mjukvaruutvecklare som vill börja arbeta med IFC-modeller.

Metod

Inledningsvis genomfördes en grundläggande litteraturstudie av BIM för att kunna finna en lämplig metod för import och export av BIM-modeller mellan ritverktyg och strukturanalysverktyg. I början av arbetet var det svårt att veta exakt vad som skulle kunna åstadkommas men efter en tids studier i ämnet stod det klart att det skulle bli möjligt att programmera en prototyp för inläsning av IFC-modeller i BRIGADE/Plus som exporterats av ritverktyget Revit Structure. Arbetets fokus har därefter legat på denna process.

Som ett första steg i detta arbete skapades, med hjälp av Revit Structure, mycket simpla IFC-modeller som fick tjäna som underlag för studien av informationsdelningsprocessen. I en grundläggande inventering av innehållet i IFC-modellen identifierades hur byggnadsmodellens olika beståndsdelar översätts från CAD-format till IFC-format. Därefter blev det också möjligt att programmera en så kallad *IFC-tolk* som gjorde det möjligt att importera IFC-modeller till BRIGADE/Plus.

Avgränsning

Arbetet har begränsats med hänsyn till LTHs specificerade omfattning av examensarbeten på civilingenjörsutbildningarna vilket innebär 30 högskolepoäng som normalt motsvarar ungefär 20 veckors arbete.

Rapporten och de teoretiska avsnitten måste betraktas som en *introduktion* i ämnet och målet för utvecklingen av IFC-tolken har i första hand varit att bevisa *möjligheten* att importera IFC-modeller till ett strukturanalysverktyg. En viktig begränsning i min prototyp är att den i huvudsak hanterar en

import av byggnadsmodellens geometri vilket inte nödvändigtvis kan jämställas med en relevant beräkningsmodell. Prototypen tillåter inte heller en export av BRIGADE/Plus-modeller till IFC-format.

2. Projekteringen av en brokonstruktion

Detta kapitel syftar till att lyfta in rapporten i ett större sammanhang genom att redogöra för projekteringsskedet av brokonstruktioner.

Broar beställs ofta av den svenska staten genom *Trafikverket.* Inledningsvis görs en omfattande förstudie i vilken en lämplig vägsträckning utarbetas med underlag från de lokala, geotekniska och miljömässiga förhållandena i det aktuella området. Om förstudien leder till beslut om att bygga bron utarbetas en lämplig utformning vilken redovisas som förslagsritningar. Dessa ritningar är underlag för entreprenörerna under anbudsskedet och kommer därefter, så snart en entreprenör valts ut, att vara underlag för att beräkna och fastställa den slutgiltiga utformningen.

Konstruktionsberäkningarna består av två delar nämligen en inledande *strukturanalys* följt av en *dimensionering*. Strukturanalysen syftar till att räkna ut kraftfördelningen på konstruktionen under belastning av trafik, egentyngd samt ”väder och vind”. Med en känd kraftfördelning är det sedan möjligt att dimensionera brons bärande element, vilket normalt innebär att räkna ut erforderlig armeringsmängd i armerade betongkonstruktioner samt erforderliga tvärsnittsmått för stålkonstruktioner enligt Eurocode. Projekteringen avslutas med att ta fram konstruktionsritningar och andra arbetshandlingar tillsammans med en uppskattning av behovet av övriga resurser som behövs för att bygga bron.

Under projekteringens gång är som sagt många olika discipliner inom byggbranschen inblandade, arkitekter, projektörer, konstruktörer och kalkylatorer spelar alla en viktig roll i laginsatsen. Mycket av informationen som utgör projekteringen är förstås disciplinöverskridande och måste kommuniceras mellan de olika lagdelarna på något sätt. Alla discipliner använder datorverktyg som är utvecklade för disciplinens specifika behov, en arkitekt använder ett ritverktyg, en konstruktör ett strukturanalysverktyg, en projektledare ett schemaläggningsverktyg etc. Disciplinspecifika applikationer är dock i regel inte kompatibla med varandra vilket ofta gör att informationsöverföringen blir ineffektiv. Det är i den här kontexten som BIM kan placeras i sitt sammanhang. Om all information istället kunde sparas i en gemensam byggnadsmodell som var tillgänglig för alla discipliner samtidigt och från vilken det gick att hämta och lägga till information på ett automatiserat sätt så skulle mycket tid sparas.

Kompatibiliteten mellan olika mjukvaror kan naturligtvis associeras med alla olika sorters applikationer och discipliner men i den här rapporten ligger fokus främst på utbytet mellan rit- och strukturanalysverktyg. Under projekteringen av en brokonstruktion kan det till dess att en slutgiltig design fastställs bli aktuellt att skicka en bromodell fram och tillbaka mellan projektörens ritverktyg och konstruktörens strukturanalysverktyg när små korrektioner utförs av respektive disciplin. Rapportens syfte är att förklara hur detta informationsutbyte kan automatiseras på ett konkret sätt.

3. Grundläggande begrepp

I detta kapitel erbjuds en förklaring till grundläggande begrepp inom ramen för BIM som bör vara bekanta för läsaren för att denne skall kunna tillgodogöra sig informationen i de kommande kapitlen.

3.1 BIM

Det finns ingen officiell definition av BIM. Olika användare världen över har anammat begreppet på lite olika sätt. I exempelvis Sverige förekommer, två olika utskrifter av akronymen, dels *Building Information Model* och dels *Building Information Modeling*. Det kan alltså vara fråga om antingen en teknik eller en sorts arbetsmetod men innebörden av att arbeta med BIM kan oavsett betraktas som densamma.

Begreppet myntades av mjukvaruleverantören *Autodesk* under lanseringen av programserien *Revit* för ett tiotal år sedan. Då hade andra mjukvaruleverantörer, exempelvis *Graphisoft* och *Bentltley Systems*, också lanserat program med liknande funktionalitet och marknadsfört dessa under begrepp som ”*Virtual Building Model*”, ”*Integrated Project Modeling*”, och ”*Project Lifecycle Management*” [8]. BIM har dock kommit att bli det i särklass dominerande begreppet för marknadsföringen av nya projekteringsverktyg och också ett uttryck för att beskriva associerade förändringar av arbetet i byggprocessen.

Att övergå till att arbeta med BIM innebär att överge de traditionella, tvådimensionella, vektorbaserade *ritverktygen* till förmån för tredimensionella, parametriserade *projekteringsverktyg*.[1)]

Traditionella ritverktyg har en mycket viktig begränsning i mängden information som kan matas in i den interna byggnadsmodellen. En sådan modell består egentligen enbart av vektorer som tillsammans visualiserar olika byggnadselement, men dessa element existerar inte i någon fysisk mening i datamodellen. Därför finns det heller inte någon möjlighet att lägga till associerad information till byggnadsdelarna, exempelvis materiella och mekaniska egenskaper. När en sådan byggnadsmodell skall användas som underlag för exempelvis en strukturanalys måste därför en stor mängd information kommuniceras vid sidan av den digitala modellen och det är upp till användaren av strukturanalysverktyget att själv sätta ihop vektorerna till avgränsade strukturelement. En BIM-modell däremot är uppbyggd på ett fundamentalt annorlunda sätt med en objektorienterad datastruktur som tillåter att byggnadselementen modelleras som enheter för vilka en godtycklig mängd enhetsspecifik information kan lagras. Denna mycket viktiga uppgradering av modelleringsförutsättningarna har lett till att BIM-modellen kan tjäna som underlag för disciplinöverskridande modellering.

Att den geometriska representationen i en BIM-modell är tredimensionell är viktigt att notera eftersom detta sätt att hantera och presentera information tydligt skiljer sig från de traditionella och mer fantasikrävande tvådimensionella pappersritningar som är det juridiskt bindande ritningsdokumentet i en konventionell byggprocess idag. Det är naturligtvis också möjligt att utvinna tvådimensionella konstruktionsritningar för olika snitt genom konstruktionen direkt ur 3D-ritningen. Komplexa konstruktioner är ibland mycket krävande att åskådliggöra i två dimensioner och för en lekman ännu svårare att begripa. Den tredimensionella visualiseringen av en konstruktion erbjuder möjligtvis en bredare målgrupp.

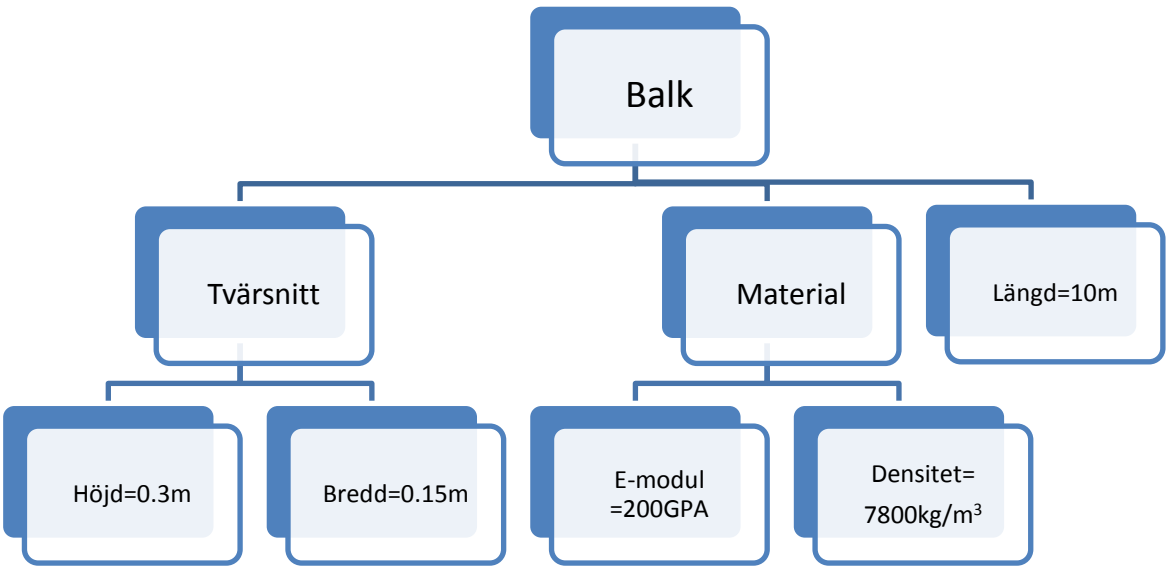
1) I rapporten används fortsättningsvis ordet *ritverktyg* istället för det kanske bättre lämpade ordet *projekteringsverktyg*, detta för att framhäva den viktigaste funktionen i applikationen i förhållande till informationsutbytet med ett strukturanalysverktyg nämligen ritfunktionen.

En BIM-modell har också en *fjärde* dimension som hanterar *projektets utveckling över tid*. Modellens geometriska beståndsdelar kopplas direkt till en aktivitetsplanering som också omfattar en resursplanering och ekonomisk kalkyl. Sammanfattningsvis blir det alltså möjligt att automatisera många tidsödande aktiviteter under projekteringen exempelvis, *mängdning*, *kollisionskontroller, arbetsscheman* och *budgetering*[9]. Modellen har förutsättningar till att bli ett aktivt dokument, inte bara under projekteringen utan under hela byggnadens livscykel och kanske inte minst under förvaltningsskedet när det blir aktuellt med reparationer och ombyggnader.

3.2 Objektorienterad modellering

Objektorienterade program och databaser är en viktig förutsättning för skapandet och hanteringen av BIM-modeller. Att bygga en modell som ett konglomerat av självständiga objekt, snarare än en enhet av vektorer, är ett sätt imitera konstruktionen som skall byggas i verkligheten. I en objektorienterad byggnadsmodell modelleras byggnadselementen ”självständigt” så att exempelvis en balk modelleras som ett *balkobjekt* och en pelare som ett *pelarobjekt* och så vidare. En byggnadsmodell kommer således, liksom i verkligheten, att bestå av flera *individuella* byggnadsdelar som kan tilldelas objektspecifik information.

Objektorientering är ett begrepp inom datavetenskapen som utgör ett speciellt arrangemang av informationen i en datamodell. Typiskt för objektorienterade datamodeller är att informationen är arrangerad i en *hierarkisk* struktur. Ett *objekt* är en avgränsad enhet av information inom denna hierarki som innehåller objektspecifika *attribut*. Ett balkobjekt kan exempelvis tilldelas attributen *längd*, *tvärsnitt* och *material*. En längd kan beskrivas med ett enda siffervärde men ett tvärsnitt och ett material kräver en mer omfattande beskrivning och istället för att belasta balkobjektet med alla dessa storheter kan tvärsnittsmått och materialegenskaper tillskrivas datamodellen i två separata underobjekt, så kallade *child objects* (I detta fall blir då tvärsnitts- respektive materialattributet *parent objects*.) På detta sätt bygger användaren upp ett hierarkiskt träd av information, nivå för nivå, anpassat efter behovet av uppdelning[7], jämför figur 1.



Figur 1. Exempel på objekthierarki för ett balkobjekt.

En annan viktig funktion i sammanhanget är *arvsfunktionen* som om den implementeras innebär att ett underobjekt tilldelas, eller *ärver*, egenskaper automatiskt från alla föräldrarobjekt högre upp i hierarkin, vilket ger en ackumulation av information nedåt genom hierarkin.

Objekt kan också tilldelas objektspecifika funktioner, så kallade *metoder*. Syftet med detta är också att dela upp informationen, att inte belasta det enskilda objektet med onödigt mycket data. I en byggnadsmodell kan metoder användas för analytiska beräkningar, en stålbalk kan exempelvis

tilldelas en metod som beskriver instabilitetsfenomen så som buckling eller vippning och en pelare kan tilldelas en metod som beskriver knäckning.

Den allomfattande definitionen av objektets attribut och metoder kan sägas utgöra en ritning, en så kallad *klass,* som kan användas för skapandet av en godtycklig mängd objekt av samma slag. Med fördefinierade klasser är det enkelt och effektivt att snabbt skapa ett stort antal olika objekt [7].

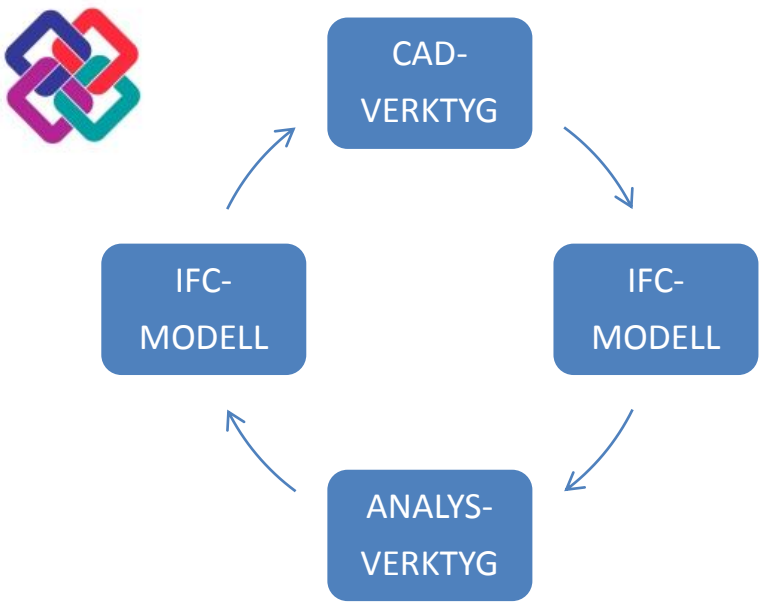
Verktyg för strukturanalys är av naturliga skäl objektorienterade, en byggnadsmodell måste av beräkningsmässiga skäl delas in i strukturelement. Det första steget i implementeringen av BIM för projekteringsarbetet inom byggbranschen är då kanske att också gå över till användandet av intelligentare, objektorienterade ritverktyg så att mjukvarorna bättre kan matchas på strukturell nivå.

3.3 IFC

Ett hinder för utvecklingen av BIM är som sagt alla kommersiella mjukvaruleverantörer som använder ett *proprietärt* filformat samt avsaknaden av en branschstandard för byggnadsmodeller vilket förhindrar kompatibilitet mellan olika sorters disciplinspecifika applikationer. I ett försök att underlätta informationsdelningen inom byggbranschen har *IAI*, *International Alliance for Interoperability* (ett internationellt konsortium skapat av kommersiella bolag och forskningsinstitut) sedan 1997 drivit utvecklingen av ett *neutralt* filformat kallat *IFC*, *Industry Foundation Classes*. Datastrukturen utvecklas med öppen källkod och kan implementeras av alla mjukvaruutvecklare i världen. Målet är att IFC skall bli standardformat för BIM-modeller inom byggbranschen [5].

IFC-modellen är avsedd att omfatta all disciplinspecifik information inom projektering, byggnation och förvaltning och är tänkt att vara ett aktivt dokument genom hela byggnadens livscykel. Detta innebär naturligtvis en stor mängd data och det är inte tänkt att mjukvaruleverantörer skall integrera hela specifikationen i sina applikationer. Istället erbjuds olika avskalade versioner, så kallade *views,* anpassade till olika informationsdelningsscenarion inom byggbranschen. Rapportens fokus ligger på vyn *IFC2*3 Coordination View*, som är utvecklad för utbyte av information mellan arkitekt- och installationsteknik- samt strukturanalysdisciplinerna. IFC-modellen utvecklas kontinuerligt och arbetet drivs sedan 2005 av IAI under arbetsnamnet *Building Smart Alliance* med hjälp av sponsorer från hela världen [5].

IFC är en objektorienterad byggnadsmodell och bygger på den så kallade *STEP*-modellen (*Standard of the Exchange of Product Model Data)* vilken används inom andra tillverkningsindustrier. Både IFC och STEP utvecklas i det ISO-certifierade programmeringsspråket *EXPRESS* som har vissa likheter med det kanske mer kända programmeringsspråket *Pascal* [2].



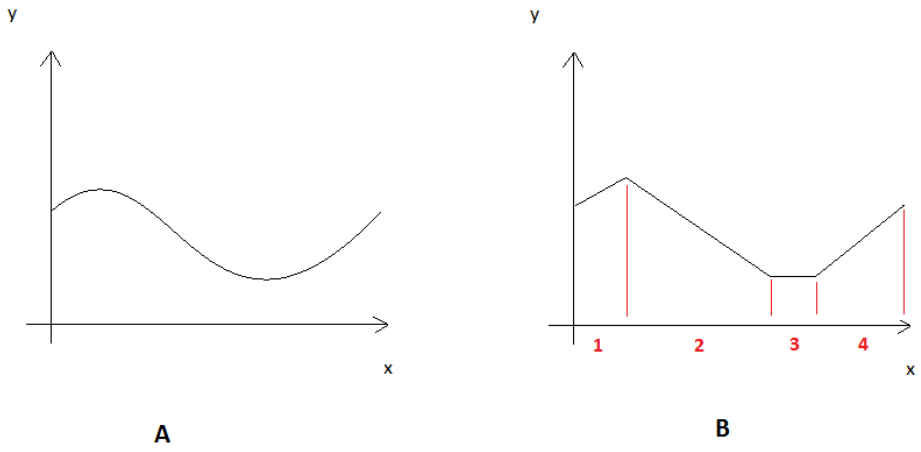
Figur 2. IFC-cykeln. Figuren visar schematiskt ett informationsdelningsscenario mellan ett CAD-vertyg och ett analysverktyg. Genom att utnyttja det neutrala IFC-filformatet kan program utrustade med en IFC-tolk importera och exportera BIM-modeller till och från andra program som stöder IFC-formatet. IFC-logotypen pryder figuren uppe i det vänstra hörnet.

3.4 Finita elementmetoden och verktyg för strukturanalys

Detta avsnitt syftar till att översiktligt beskriva strukturanalysen och strukturanalysverktyget för att ge läsaren en känsla för vilken sorts information som möjligtvis skulle kunna importeras till detta från exempelvis ett ritverktyg.

Att genomföra en strukturanalys av en konstruktion innebär att räkna ut hur krafterna kommer att fördelas och bäras av konstruktionen vid de mest ogynnsamma lastställningarna som kan tänkas uppkomma under byggnadens livslängd. Strukturanalysen av en brokonstruktion är en mycket omfattande simulering som är besvärlig att bemästra analytiskt, istället kan en *numerisk metod* tillämpas som ger en *approximativ* lösning.

Finita elementmetoden, eller *FEM*, är ett kraftfullt matematiskt verktyg som tillämpas inom alla sorters ingenjörs- och forskningsområden. Namnet kan härledas ur en viktig funktion i den numeriska metoden som tillåter att det analyserade objektet delas in i mindre, *finita,* delområden eller *element* för att underlätta beräkningen. Poängen med uppdelningen är att även om en variabel varierar på ett mycket komplicerat och olinjärt sätt inom hela det analyserade systemet så är det ofta en god approximation att anta att variationen inom ett litet begränsat delområde varierar på ett betydligt mindre komplicerat sätt, exempelvis linjärt eller kvadratiskt vilket erbjuder en analytisk lösning lokalt[1], se figur 3.



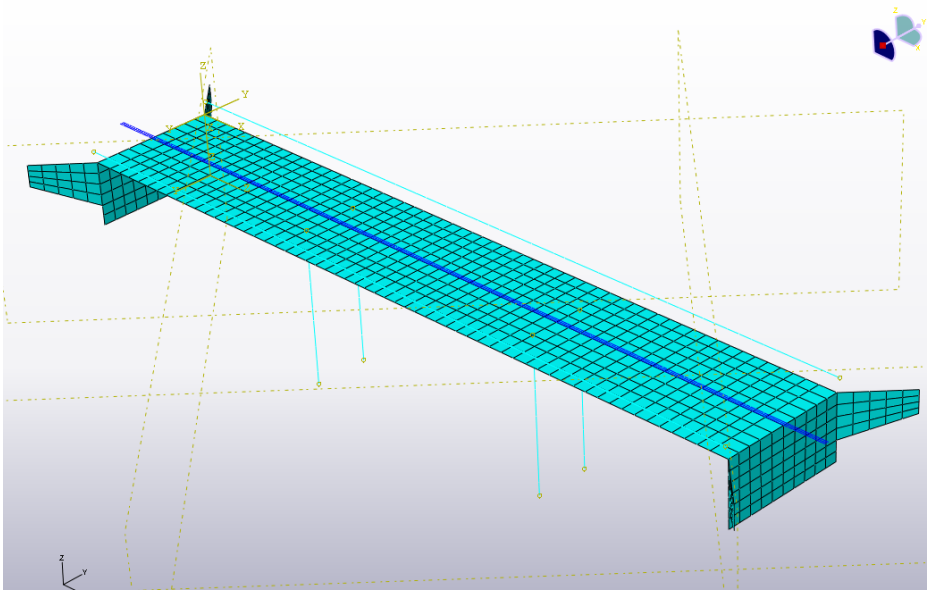
Figur 3. Exempel på skillnaden mellan en exakt och en approximativ FEM-formulering. I figur A varierar en variabel, y, olinjärt över x-axeln. I figur B har x-axeln delats in i fyra delområden inom vilka y varierar linjärt.

Ett större område delas således in i mindre delområden, eller *element*. Varje element tilldelas en approximativ ekvation och pusslas sedan samman med andra element enligt vissa regler till ett sammanhängande område, ett så kallat *mesh* (efter engelskans ord för rutnät). Ett element spänns upp av ett antal så kallade *noder,* punkter i elementet som tillskrivs variabla egenskaper eller *frihetsgrader*, exempelvis *translation* och *rotation*.

Finita elementmetoden kan användas för att lösa både randvärdesproblem och begynnelsevärdesproblem. Inom strukturmekaniken har problemen vanligtvis karaktären av

randvärdesproblem i två eller tre dimensioner där vissa storheter längs med randen av det studerade området är kända, exempelvis upplagsvillkor och laster.

Generellt gäller såklart att approximationen blir bättre ju mindre delområdena är och ju högre polynomrang den lokala approximationen tilldelas vilket innebär fler frihetsgrader och fler ekvationer i systemet. I verkliga tillämpningar innehåller ett typiskt system tusentals frihetsgrader och måste därför lösas med hjälp av datorkraft [1]. Strukturanalysverktyg baserade på FEM är således program med vars hjälp det går att upprätta en FEM-modell samt räkna ut en approximativ lösning till det aktuella ekvationssystemet. Programmet har vanligtvis tre särskilda funktioner, dels en *preprocessor* som skapar ett beräkningsbart mesh, se figur 4, en *solver* som löser ekvationerna samt en *postprocessor* som använder resultaten för att presentera efterfrågad information.



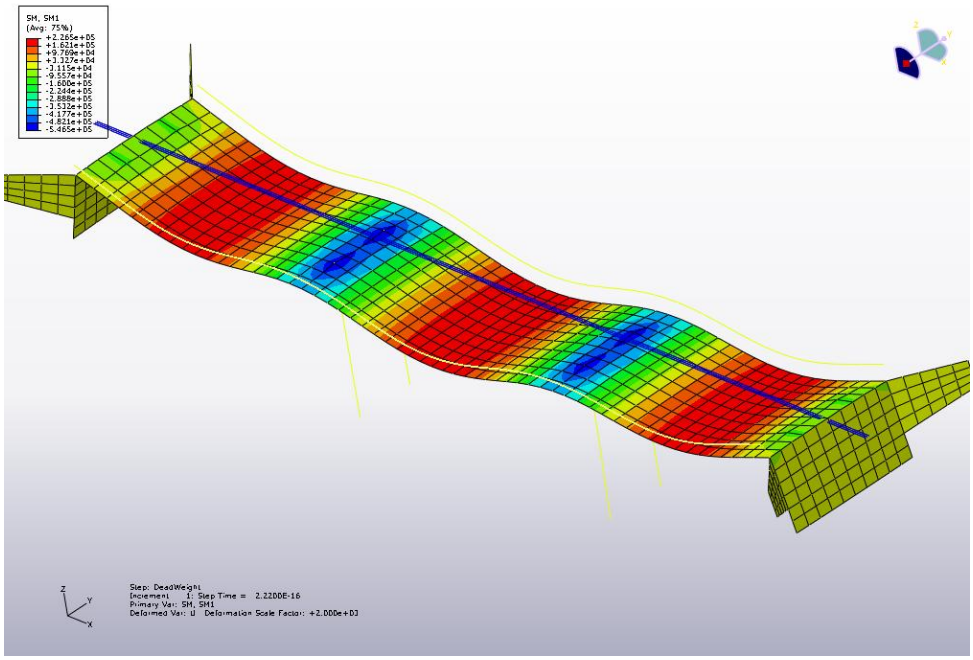
Figur 4. Bilden visar ett exempel-mesh i BRIGADE/Plus preprocessor bestående av rektangulära element. Modell av en järnvägsbro med stödmurar, vingmurar, pelare och broplatta samt räler.

Preprocessorn och postprocessorn är ofta integrerade i ett grafiskt gränssnitt vilket tillåter användaren att modifiera modellen på ett enkelt sätt samt studera resultaten i, för FEM-verktyg kännetecknande, färgkodade tredimensionella bilder, se figur 5, med tillhörande diagram och tabeller.

Normalt baseras analysen av brokonstruktioner på linjärelastiska beräkningsmodeller. Den grundläggande ekvationen för sådana problem är som bekant *Hookes lag* som ger att spänningen, σ, i ett material är lika stor som elasticiteten, E, multiplicerad med töjningen, ε.

$$\sigma = E\,\varepsilon \qquad (2.3.1)$$

Dessa formuleringar avser ett *perfekt elastiskt* material. I verkligheten är dock exempelvis, armerade betong- och stålkonstruktioner inte perfekt elastiska eftersom betong spricker och stål plasticerar vid en viss spänningsnivå. Enligt svenska normer studeras normalt endast linjärelastiska bromodeller vilket innebär att man vid dimensionering endast tillåter en belastning under *sträckgränsen* [3].)



Figur 5. Visualiserat beräkningsresultat i BRIGADE/Plus postprocessor. Bilden visar järnvägsbron i figur 4 som belastats med egentyngd. Deformationer på bron har fått en kraftig uppskalning. Röd färg markerar positivt fältmoment, blått markerar negativt stödmoment.

Syftet med strukturanalysen av en brokonstruktion är som sagt att ta fram de dimensionerande snittkrafterna, det vill säga *normalkraft*, *tvärkraft* och *moment,* eftersom det är dessa storheter som ligger till grund för beräkningen av erforderlig armeringsmängd och tvärsnittsarea. Snittkraften räknas som en integration av spänningarna över tvärsnittets yta och denna enhetsomvandling måste ske i något steg av beräkningen. I en förenklande diskretisering av spänningsekvationerna reduceras bromodellens byggnadselement till ett fåtal olika sorters elementära strukturelement exempelvis *balkar*, *pelare*, *plattor* och *skivor* vilka uttrycks på ett matematiskt sätt med *stång-*, *balk-* och *platteori*. Poängen är att beräkningsresultatet för sådana element erhålls direkt i de sökta enhetstyperna *kraft* och *moment* eftersom integrationen över tvärsnittet är ”inbyggd” i omformuleringen. Den grundläggande FEM-ekvationen kan formuleras på ett enkelt sätt: om man belastar en konstruktion styvheten *K* med kraften *f* kommer detta att ge upphov till en deformation motsvarande *a*.

$$K\,a = f \qquad (2.3.2)$$

Styvhetsmatrisen K innehåller strukturelementens geometriska mått så som längder, tvärsnittmått och tröghetsmoment samt materialegenskaper exempelvis elasticitetsmodul, tvärkontraktionstal och längdutvidgningskoefficient etc. beroende på vilket fysiskt fenomen som studeras. Förskjutningsvektorn a och kraftvektorn f innehåller alla okända variabler men också begynnelsevärdena. Förskjutningsvektorn tillskrivs alla låsta frihetsgrader vid upplag, i byggsammanhang ofta av enhetstyperna meter eller radianer. Kraftvektorn f tillskrivs alla kända belastningar på konstruktionen [1].

Sammanfattningsvis består den ingående informationen i en FEM-modell av geometri, materialegenskaper och laster.

4. Metodik

I det här kapitlet presenteras en metodik för export av BIM-modeller till IFC-format. Rapportens fokus är i första hand riktat mot möjligheterna att exportera och importera BIM-modeller mellan ritverktyg och verktyg för strukturanalys men mycket av informationen i kapitlet gäller naturligtvis också för utbytet av information mellan andra sorters disciplinspecifika applikationer.

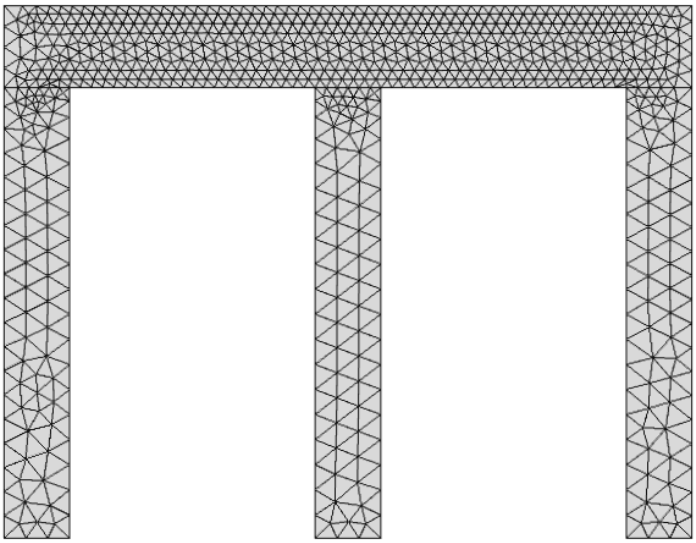
4.1 Från konstruktionsritning till beräkningsmodell – eller tvärtom?

Även om det inte är möjligt att genomföra ritningsarbetet och strukturanalysen exakt samtidigt så är det såklart fördelaktigt om det disciplinöverskridande arbetet kan ske någorlunda parallellt så att eventuella fel eller brister kan korrigeras på ett tidigt stadium under projekteringen. Import och export av BIM-modeller mellan disciplinspecifika mjukvaror måste därför fungera med endast minimal handpåläggning för användaren. I många strukturanalysprogram är det möjligt att importera en CAD-ritning. I BRIGADE/Plus är det exempelvis möjligt att läsa in filer av *DWG*-format men en byggnadsmodell i detta format består dock endast av uppritade vektorer. Användaren måste själv dela upp och sätta ihop vektorerna till lämpliga strukturelement samt infoga kopplingar mellan dessa för att fullborda omvandlingen till beräkningsmodell. Detta är ett tidskrävande arbete och medför en risk för missuppfattning vad gäller konstruktionens prestanda om kommunikationen mellan projektör och konstruktör är bristfällig. En intelligentare BIM-modell skall därför, om möjligt, exporteras på ett betydligt smidigare sätt.

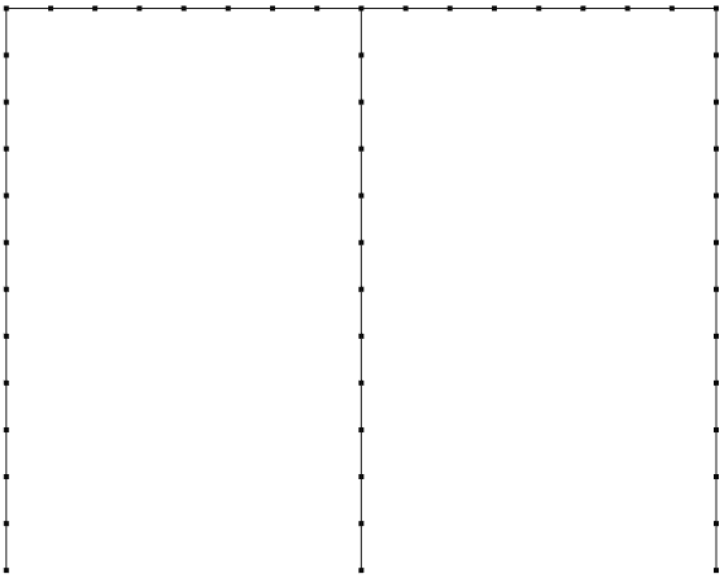
Inledningsvis: vad är skillnaden mellan en konstruktionsritning och en beräkningsmodell?

Det finns fem grundläggande aspekter att ta hänsyn till vid omvandlingen från konstruktionsritning till beräkningsmodell:

1. *Geometri* – En konstruktionsritning är en skalenlig visualisering av verkligheten och en instruktion till den som skall bygga byggnaden. Därför är detaljrikedomen mycket stor med en precision på millimetern när. En beräkningsmodell däremot är, liksom de matematiska ekvationerna, ofta en grov förenkling av verkligheten vilket innebär att modellens detaljrikedom måste skäras ned för att kunna passa in i en rimligt omfattande strukturanalys. I en beräkningsmodell där strukturelementen är av typen balk- och plattelement etc. visualiseras dessa ofta som streck eller tvådimensionella månghörningar och denna fysiska reducering av ritningen är viktig därför att den påverkar FEM-modellens *mesh*, dvs. antalet element och frihetsgrader och därmed storleken på ekvationssystemet och beräkningstiden. Figur 6 och 7 illustrerar den uppenbara skillnaden mellan en solidmodell och en balkmodell.



Figur 6. Tvådimensionell solidmodell av en ramkonstruktion. Pelare och balk har en skalenlig geometri. Modellen består av en sammanhängande linjärelastisk kropp. Totalt 1433 element, 2729 frihetsgrader.



Figur 7. Balkmodell av samma ramkonstruktion. Geometrin har reducerats till tunna streck. Modellen består av fyra balkar. Totalt 52 element, 159 frihetsgrader.

2. *Urskiljning och uppdelning av strukturelement* – En byggnadsmodell består normalt av ett stort antal olika byggnadselement med olika material, geometri och strukturmekaniska funktioner. Uppdelningen av modellen i strukturelement är viktig eftersom den frilägger alla byggnadselement som påverkar konstruktionens bärförmåga (I figur 6 och 7 finns det exempelvis fyra olika strukturelement, en balk och tre pelare). Vidare skall endast de delar av den geometriska ritningen som på ett avgörande sätt påverkar strukturanalysen, genom sin bärförmåga och eller genom sin belastande egentyngd, ingå i beräkningsmodellen. Alla

andra geometrier skall plockas bort. I bilaga 8.1 finns en generell beskrivning av brokonstruktioners geometri och strukturella uppdelning.

3. *Kopplingar* – Byggnadselement i en konstruktion kan *överföra* krafter mellan varandra. På en ritning är det inte alltid lätt att enkelt dra slutsatser kring kopplingarnas avsedda funktionalitet. Det som syns på en ritning är det som syns i verkligheten vilket inte nödvändigtvis visar på konstruktionens strukturmekaniska egenskaper. Likväl måste kopplingen i noderna mellan två strukturelement definieras på ett tydligt sätt för att konstruktionens prestanda och beteende under belastning skall simuleras på ett korrekt sätt. I en beräkningsmodell skiljer man på kopplingar med *fritt upplag, fast inspänning och fjädrande upplag.* Dessa tre sorters upplag kan också varieras genom att låsa eller låsa upp en eller flera frihetsgrader.

4. *Materialegenskaper* - Den grundläggande ekvationen (2.3.1) bygger på det elastiska sambandet mellan belastning och töjning av ett material. I en beräkningsmodell måste således alla strukturelement tilldelas materialstorheter så som *elasticitetsmodul*, *tvärkontraktionstal* och *längdutvidgningskoefficient*.

5. *Laster* - En konstruktionsritning säger ingenting om vilka laster som kommer att verka på konstruktionen men i en beräkningsmodell skall aktuella laster och geometri kopplas ihop. Laster delas normalt in i flera kategorier beroende på hur de angriper på konstruktionen exempelvis *punktlast*, *linjelast* eller *ytlast*.

En naturlig fråga i sammanhanget bör vara: vem gör vad och när i ett disciplinöverskridande samarbete? I det här fallet är det möjligt att tänka sig två olika informationsdelningsscenarion:

1. CAD-projektören tar första steget och ritar upp en geometri för export till konstruktörens analysverktyg.
2. Konstruktören tar första steget och genomför en strukturanalys varefter den testade geometrin exporteras till CAD-projektören för detaljutformning. (Det finns naturligtvis också ingenjörer som *både* ritar och räknar.)

I båda fall är det därefter möjligt att tänka sig en parallell utveckling med en informationsdelning åt båda håll men det är intressant att föreställa sig hur mycket information som kan skapas redan i det första steget innan den första exporten genomförs.

I det första scenariot är det teoretiskt möjligt att hela beräkningsmodellen, inklusive koppling- och lastfördelning, skapas av projektören i projekteringsverktyget vilket innebär minimalt jobb för konstruktören men eftersom mycket av den sortens arbete normalt inte ingår i arkitektens/projektörens traditionella arbete är det tveksamt om denna arbetsfördelning är att föredra. Det är kanske mera rimligt att i detta fall sträva efter en genomtänkt och för strukturanalys väl anpassad geometrisk export så att den manuella handpåläggningen efter import blir minimal.

I det andra scenariot måste konstruktören ha en geometri att utgå ifrån när beräkningsmodellen skall utformas. Om konstruktören utarbetar ett första geometriskt förslag är detta möjligtvis ett bra utgångsläge för projektet eftersom konstruktionen analyserats redan innan konstruktionsritningen ritas upp. En uppenbar nackdel är dock att en beräkningsmodell inte är lika detaljerad som en CAD-modell vilket skulle kunna vara av betydelse för vissa detaljfrågor.

En informationsdelning ställer som sagt höga krav på tekniken och applikationens förmåga att skriva ut och tolka in data på rätt sätt. Det är svårt att veta till vilken grad informationsdelningen kan komma att utvidgas till. Det finns ett informationsglapp mellan konstruktionsritning och beräkningsmodell, kanske krävs det trots allt att en mänsklig hjärna först tolkar den importerade modellen och gör vissa kvalificerade korrektioner för att hela omvandlingsprocessen skall bli korrekt. Kanske är BIM-modellens största styrka i det här sammanhanget delning av sådan information som är gemensam för konstruktionsritning och beräkningsmodell dvs. geometrin.

4.2 Informationsdelning mellan olika applikationer

Det finns inget standardformat för byggnadsmodeller i byggbranschen och eftersom olika mjukvaruleverantörer använder proprietära filformat i stor utsträckning är det ibland omöjligt att överföra en modell från en applikation till en annan genom informationsdelning. Undantag finns och det är såklart möjligt för två olika leverantörer att tillsammans utveckla ett filformat som båda två kan hantera i sina respektive program men så länge kundernas efterfrågan på en öppnare informationsdelning inte kan motiveras med en motsvarande lönsamhet för denna tekniska utveckling är utsikterna förmodligen små för den här sortens initiativ. Samtidigt blir det inom andra områden av IT-världen allt vanligare att jobba enligt *open source*-konceptet, det vill säga att programmera med öppen källkod vilket öppnar för användaren att själv modifiera programmets funktioner efter eget behov. En alternativ lösning på informationsdelningsproblemet kan vara att mjukvaruleverantörerna implementerar stöd för ett neutralt filformat som gör det möjligt att *spara* och *öppna* en byggnadsmodell i ett format som är helt oberoende av mål- eller mottagarapplikation.

Olika neutrala filformat har utvecklats i syfte att underlätta informationsdelningen av byggnadsmodeller, exempelvis *IFC*, *ODBC* och *XML*. IFC-formatet, med filsuffixet ”*.ifc”,* sägs vara fördelaktigt på grund av sin kompakta filstorlek i kombination med det faktum att filen kan öppnas och läsas utanför de adresserade applikationerna som en helt vanlig textfil vilket medger en enkel granskning av uppburen data. IFC-modellen bygger på en öppen specifikation som publiceras och uppdateras på internet av Building Smart Alliance [5].

För att ett program ska vara IFC-kompatibelt krävs en funktion som gör det möjligt att importera och exportera IFC-modeller. Detta betyder att programmet måste kunna göra en intern, modifierbar modell av informationen i IFC-filen och samtidigt kunna översätta den interna modellen till IFC-format. För att kunna åstadkomma detta måste mjukvaruutvecklaren programmera en så kallad *parser,* en tolk som kan tolka informationen mellan IFC-syntaxen och programmets egen syntax. Varje applikation måste utrustas med en egen, skräddarsydd parser som kan exportera och importera IFC-modeller. I kapitel fem presenteras en *parser-prototyp* som har skapats i syfte att försöka möta examensarbetets uppställda mål.

Som en marknadsföring för mjukvaruleverantörer och för att underlätta för kunder som önskar att arbeta i en modelleringsmiljö som hanterar IFC-formatet har vissa applikationer tilldelats status som *IFC-kompatibel*. Denna status uppnås först efter en IFC-certifieringsprocess i vilken parsern testas på ett standardiserat sätt både för export och för import. Det finns över 100 mjukvaruleverantörer i världen som hävdar att de har stöd för IFC [10]. I bilaga 8.2 finns en lista på alla certifierade applikationer i juli 2011.

4.3 IFC-modellen

IFC-modellen är den enskilt viktigaste biten i informationsdelningspusslet mellan olika byggrelaterade mjukvaror. Tanken är att den skall kunna användas som ett gemensamt dokument för alla discipliner inom byggbranschen genom hela byggnadens livscykel, från projektering till rivning. Byggnadsmodellen är utformad för att kunna arkivera en riklig mängd information. Det är därför inledningsvis viktigt att poängtera att modellen förutom det mest uppenbara innehållet av olika byggnadsdelar och installationer också kan innehålla mer abstrakt och på andra sätt värdefull information så som aktivitetsplanering, mängdning, budgetering etc. beroende på vilken IFC-vy som användaren väljer att arbeta med [5].

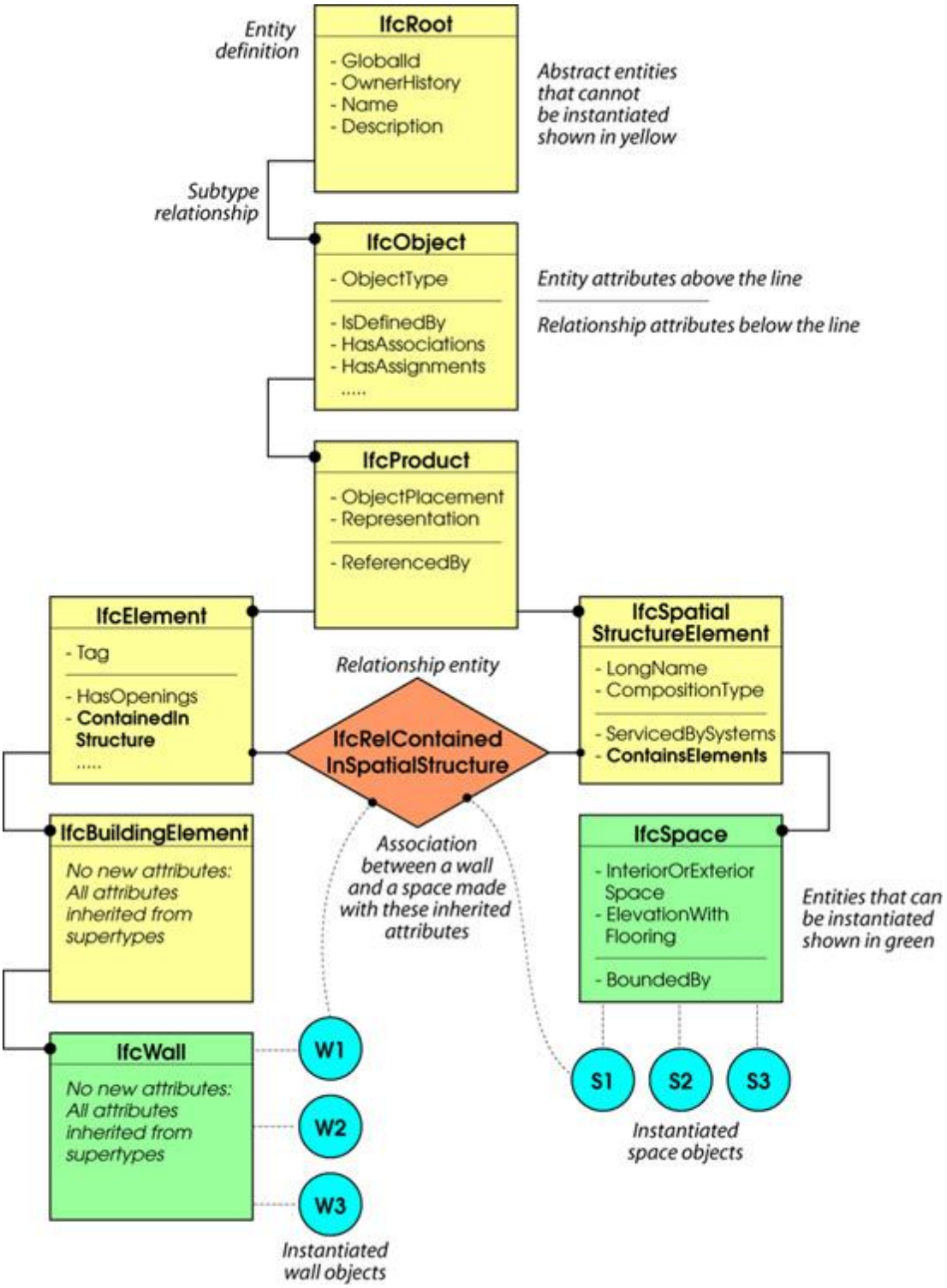
IFC-modellen är organiserad i en hierarki enligt figur 8. På en översiktlig skala är ”objektträdet” indelat i fyra övergripande nivåer, så kallade *Layers*. Varje nivå innehåller ett antal olika avdelningar, så kallade *schemas*, i vilka objekten lagras som så kallade *entities* eller *enheter.* Det hierarkiska upplägget i modellen har en viktig innebörd, en enhet får endast relatera eller referera till en enhet på samma nivå eller en nivå under. Detta innebär att en enhet i ett undre lager, inom EXPRESS-språket en så kallad *subtype,* eller *subenhet,* kan användas av flera olika enheter i ett lager ovanför, så kallade *supertypes* eller *superenheter.* Denna begränsning skall verka för att underlätta informationsdelningen mellan olika disciplinspecifika applikationer som arbetar i olika övre lager men som också använder gemensamma resurser i de undre lagren[4]. IFC-modellens fyra nivåer kan kortfattat beskrivas enligt:

1. *Domain Layer* – Den högsta nivån består av disciplinspecifika scheman med information som bara används inom just denna disciplin. För *Structural Engineering* innebär det exempelvis enheter med koppling till grundläggningen så som fundament och pålar medan det för exempelvis *HVAC,* (*Heating, Ventilation, Air Conditioning*, eller på svenska ungefär *VVS*), kan handla om fläktar, ventilationstrummor och avloppsenheter etc.
2. *Interoperability Layer* – I den andra nivån återfinns element och koncept som är disciplinöverskridande och delas av olika disciplinspecifika applikationer. Avdelningen *Shared Building Elements* innehåller exempelvis enheter för balkar, pelare, plattor, väggar, dörrar, fönster etc.
3. *Core Layer* – Den tredje nivån innehåller avdelningar som knyter ihop enheterna i de högre nivåerna. Avdelningen *Kernel* definierar grundläggande koncept exempelvis beroenden mellan olika enheter, processer, inblandade aktörer osv. Avdelningen *Product Extension* definierar innehållet i ett projekt, en byggnad, en våning, ett rum osv.
4. *Resource Layer* – Den lägsta nivån innehåller den mest konkreta informationen bestående av geometri, material, tidsangivelser, kostnader och olika beskrivningar. Enheterna i den här nivån är oberoende av förhållanden och kan därför användas av flera olika projekt, byggnader, våningsplan osv. Det räcker således med att definiera en balkgeometri en gång i denna nivå om exakt samma typ av balk skall användas på flera olika platser i byggndsmodellen[4].

Figur 8. IFC-modellens objektstruktur. Översta lagrets avdelningar markerade med cirklar, understa lagret markerat med åttahörningar. I figuren har avdelningar som ingår i vyn IFC 2x3 markerats med grön färg. Figur från [6].

En vy innehåller som sagt enbart en delmängd av hela specifikationen. Det finns 653 olika enheter fördelade på 17 scheman i vyn IFC2*3 Coordination View 2.1 [6].

Det är i den här rapporten inte meningsfullt att redogöra för allihop men IFC-modellen kanske enklast förklaras genom ett beskrivande exempel (inspirerad av artikeln [4]). Figur 9 är en hierarkisk karta som visar hur två grundläggande enhetstyper i ett hus byggs upp genom att *ärva* information från superenheterna som finns högre upp i hierarkin. Figuren visar också på en annan viktig funktion i IFC-modellen nämligen hur de två elementen relaterar till varandra. De två härledda enheterna är dels väggenheten *IFCWALL* som är en fysisk byggnadsdel, och dels rumsenheten, *IFCSPACE* som används för att definiera ett rum eller en yta för exempelvis energiberäkningar eller för olika definitioner inom fastighetsförvaltningen.



Figur 9. Strukturell karta över en IFC-modell bestående av väggarna W1,W2 och W3 som tillsammans utgör rummet S1. Abstrakta enheter markerade i gul färg, explicita i grön och röd. Figur från [4].

Alla element i *Resource*-lagret utgår från enheten *IFCROOT*, belägen i *Core-lagret*, i avdelningen Kernel, se figur 8. Detta är en så kallad *abstrakt* enhet vilket betyder att den endast existerar i den mall som måste användas av den exporterande applikationens parser när den gör översättning från intern modell till IFC-modell. En abstrakt enhet skrivs inte ut som en textrad och finns inte i den exporterade IFC-filen. I figur 9 är alla gulmarkerade enheter abstrakta, endast IFCWALL, IFCSPACE och IFCRELC är *explicita* enheter som faktiskt syns i IFC-filen.

IFCROOT har som synes fyra attribut nämligen *GlobalID, OwnerHistory, Name* och *Description.* Det första är ett identifikationsattribut, varje objekt i IFC-modellen skall ha en unik uppsättning tecken som skall säkra en effektiv arkivering och åtkomst av objektet. OwnerHistory används för att beskriva vem som har skapat enheten, i vilket program detta gjorts samt vem som senast gjort en ändring av enheten med tillhörande datum. Dessutom finns två *valfria* (*OPTIONAL*) attribut nämligen *Name* och *Description*, som kan implementeras vid behov [6]. Många attribut i IFC-specifikationen är just valfria vilket skall öka modellens flexibilitet men det är viktigt för en mjukvaruutvecklare att notera den här egenskapen eftersom detta innebär att vissa attributs vara eller icke-vara beror på exporterande applikation.

IFCROOT har tre olika typer av möjliga subenheter nämligen *IFCPROPERTY, IFCRELATIONSHIP eller*, som i detta fall, *IFCOBJECT.* Den senare enheten skall representera alla konkreta ting i en byggnadsmodell, förutom byggnadsdelar också kostnader, aktiviteter, personer etc. IFCOBJECT ärver de fyra attributen från superenheten IFCROOT men har också sex egna attribut. Av dessa är *ObjectType* det enda *explicita* attributet. ObjectType är en textsträng som skall beskriva vilken sorts objekt det handlar om. I figur 9 är explicita attribut uppställda högst upp och avdelade med ett streck från de övriga attributen i enheternas attributlista. Attribut av typen *INVERSE,* i detta fall *IsDefinedBy, HasAssociations, HasAssignments* osv. är så kallade ”*omvända tvärreferenser”* mellan enheter som aldrig skrivs ut i IFC-filen. Inte heller attribut av typen *DERIVED,* som är beräkningar av explicita attribut, finns med i IFC-textfilen. Eftersom det inledningsvis är mest intressant att förklara innehållet i en IFC-fil ges i fortsättningen endast en förklaring till explicita attribut.

IFCOBJECT har sju olika sorters subenheter i vilka objektets egenskaper kan definieras. För objekt som har en rumslig utbredning skall geometrin beskrivas med hjälp av enheten *IFCPRODUCT*. Enheten har två explicita attribut, *ObjectPlacement* och *Representation.* Placeringsattributet är en referens till objektets placering som är antingen *absolut* och då refererar till det globala koordinatsystemet som gäller för hela modellen eller *relativt, en* referens till ett lokalt koordinatsystem (alt. *constraint,* i förhållande till en annan byggnadsdel). Attributet *Representation* är en referens till objektets geometriska representation. Vid behov kan ett objekt representeras på flera olika sätt med olika geometrier (på samma position), detta för att öka flexibiliteten i modellen. Som tidigare påpekat finns det ofta geometriska skillnader mellan en konstruktionsmodell och en beräkningsmodell och här finns alltså en möjlighet att rymma båda två, jämför solidmodell i figur 6 och balkmodell i figur 7.

Nästa nivå i hierarkin delar hierarkin i två förgreningar, en för väggarna och en för rummet. *IFCELEMENT* skall beskriva alla komponenter i en så kallad *AEC product* (*Architechture, Engineering, Construction*). Det behöver inte nödvändigtvis handla om balkar, pelare, fönster, installationer osv. utan kan också vara element av typen *void elements,* dvs. ursparingar och hål men alltså inte enheter som beskriver ett rum eller en yta. Enheten har bara ett explicit attribut nämligen *Tag,* som är ett identifikationsnummer.

IFCBUILDINGELEMENT är, precis som rubriken antyder, en enhet som beskriver en viktig del av byggnaden, alltså både strukturella och arkitektoniska byggnadsdelar. Enheten har inga egna explicita attribut (dock 12 attribut av typen INVERSE).

IFCWALL beskriver en vertikal konstruktion, med avgränsande och alternativt lastbärande funktion. Enheten ärver totalt åtta explicita attribut, från de första fyra i IFCROOT ner till det sista i IFCELEMENT. I en IFC-fil skrivs enheten ut som en textrad enligt exemplet nedan, de åtta attributen är åtskilda med kommatering:

```
#100=IFCWALL('0Ki3Hm7wXDTQ1NH2W5',#33 ,Wall 1, ' Load bearing wall-
Concrete', 'IfcWall',#82,#153,'187543');
```

(I avsnitt 3.4 ges en förklaring till IFC-filens syntax samt en vidareutveckling av hierarkin nedåt genom Resource-lagret.)

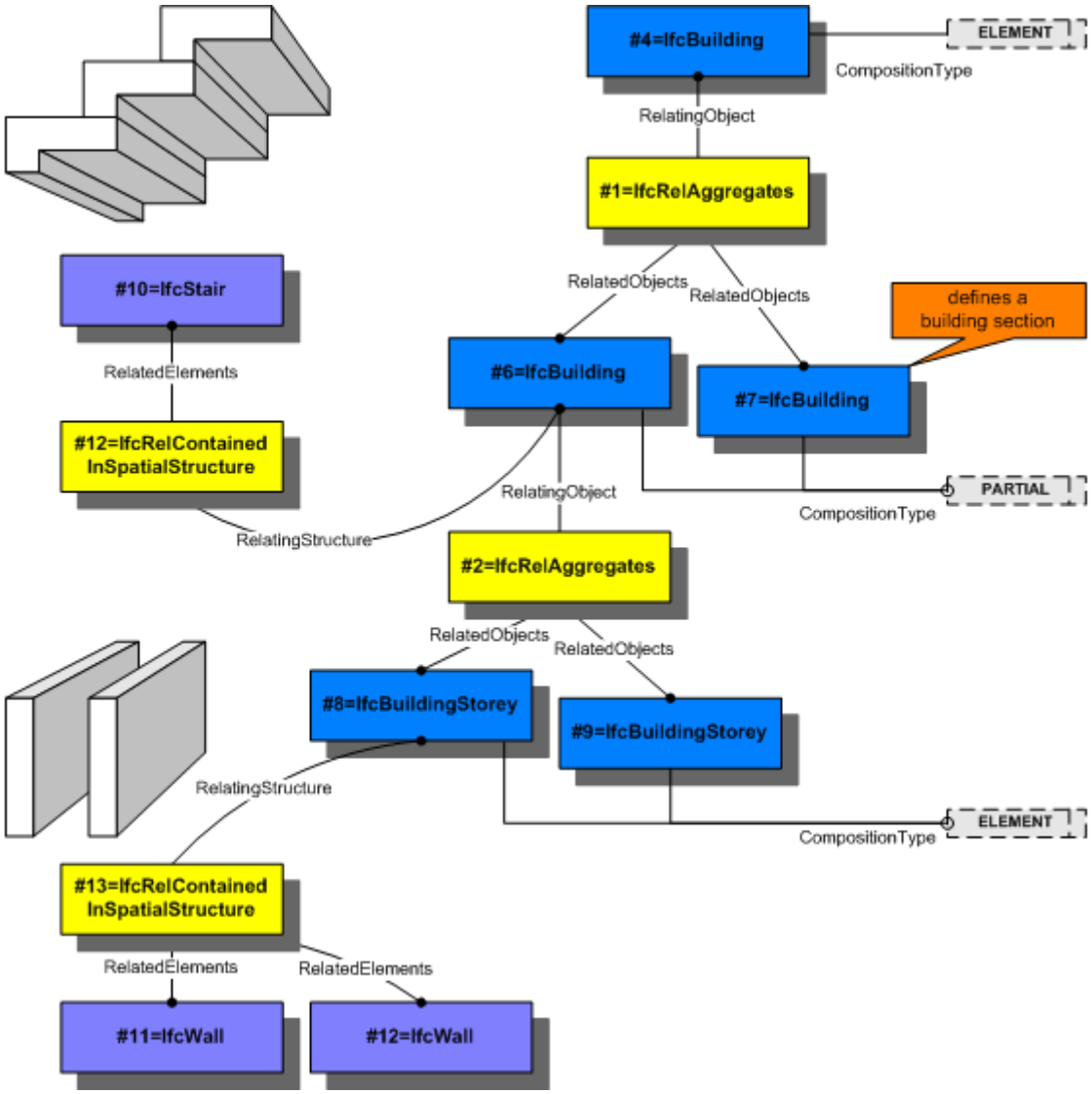
I den andra grenen är den översta enheten *IFCSPATIALSTRUCTUREELEMENT,* som definierar ett rum. (Denna enhet är en subenhet till en av fyra möjliga superenheter i en rumslig hierarki om fyra nivåer med fallande skala enligt; *IFCSITE, IFCBUILDING, IFCBUILDINGSTOREY* och *IFCSPACE*. Endast enheterna IFCBUILDING och IFCBUILDINGSTOREY är obligatoriska enheter.) IFCSPATIALSTRUCTUREELEMENT har två explicita attribut, *LongName,* för namngivning samt *CompositionType,* som anger huruvida elementet är av typen *COMPLEX, ELEMENT* eller *PARTIAL,* dvs. en sammansättning av flera rum, ett singelrum eller en del av ett rum [6].

IFCSPACE har två explicita attribut, *InteriorOrExteriorSpace* anger huruvida rummet är utvändigt eller invändigt i förhållande till byggnaden och *ElevationWithFlooring* anger på vilken höjd golvnivån finns i rummet (medelvärde om golvet lutar). Enheten ärver därutöver nio explicita attribut från sina parent-objekt. I en IFC-fil kan enheten se ut enligt exemplet nedan:

```
#165=IFCSPACE('UKd3Hm2wXVHY1Nc2W4',#33 ,Space 1, 'Small office
space', 'IfcSpace',#166,#179, 'Space 1-', 'ELEMENT', 'INTERNAL',
'50');
```

Relationen mellan väggen och rummet bokförs i enheten *IFCRELCONTAINEDINSPATIALSTRUCTURE* som ligger på samma nivå som IFCROOT, dvs. i *Core*-lagret, under avdelningen Product Extension. Enheten har två attribut dels *RelatedElements,* en lista över relaterade element och dels *RelatingStructure* som anger i vilket rum enheterna finns [6]. I detta exempel skapas relationen mellan väggarna och rummet genom att bokföra *W1-3* i attributet RelatedElements och rummet *S1* i attributet RelatingStructure.

Figur 10 ger en utökad förklaring till relationshanteringen mellan olika element i IFC-modellen. Olika rum och byggnadsdelar kopplas ihop genom användningen av relationsenheter, förutom IFCRELCONTAINEDINSPATIALSTRUCTURE också *IFCRELAGGREGATES* som definierar en sammansättning av flera rum.



Figur 10. Relationskarta. Trappan med enhetsnummer #10 kan spåras uppåt till byggnad #6 som i sin tur är ett aggregat till byggnaden #4. I byggnad #6 finns två våningsplan, #8 och #9 varav det första innehåller två väggar #11 och #12. Jämför figur 9. Figur från [6].

En annan viktig funktion i IFC-modellen är så kallade *property sets*, en uppsättning enheter som gör det möjligt att lagra också sådan godtycklig information som inte specificerats i den aktuella IFC-vyn. På detta sätt förbättras IFC-modellens flexibilitet avsevärt jämfört i det fall modellen endast kunde lagra information som kan definieras av de specificerade enheternas attribut. I sammanhanget kan det påpekas att alla länder har olika byggnadsnormer vilket på förhand gör det svårt att författa en internationell standard för byggnadsmodeller. Denna funktion kan möjligtvis överbrygga eventuella informationsgap i IFC-specifikationen [4]. Enheterna *IFCSINGELPROPERTYVALUE, IFCLISTPROPERTYVALUE* och *IFCTABLEPROPERTYVALUE* har attribut som lagrar ett värde, en lista med värden respektive en tabell med värden. En godtycklig mängd sådana subenheter kan ingå i superenheten *IFCPROPERTYSET,* placerad i Resource-lagret. Kopplingen av ett property set till ett visst objekt bokförs i enheten *IFCRELDEFINESBYPROPERTIES* [6].

4.4 IFC-syntaxen

I föregående avsnitt gavs en kort introduktion till IFC-modellens struktur. I exemplet med väggarna och rummet gavs en förklaring till enheternas abstrakta superenheter men som den intresserade läsaren förstår återfinns den konkreta informationen, exempelvis objektens geometriska mått och dylikt, längre ner i hierarkin. Det här avsnittet är en vidareutveckling av IFC-hierarkin och precis som tidigare är underlaget ett konkret exempel, i det här fallet en stålbalk. Dessutom skall avsnittet också förklara hur IFC-syntaxen ser ut, det vill säga hur texten skall läsas och förstås i en IFC-fil.

IFC-filen är som sagt en ren textfil som är arrangerad enligt det så kallade *STEP-physical file format*. Det finns två avdelningar, en *HEADER SECTION* och en *DATA SECTION.* Header-sektionen sträcker sig bara över ett fåtal rader och innehåller endast en presentation av aktuell vy och avsändarapplikation med tillhörande versionsnummer enligt exemplet nedan som har skapats för vyn *Coordination View 2.1* med *Autodesk Revit Structure 2011*.

```
HEADER;
FILE_DESCRIPTION(('ViewDefinition [CoordinationView]'),'2;1');
FILE_NAME('Project Number','2011-04-13T16:26:57',(''),(''),'Autodesk Revit Structure 2011 -
1.0','20100903_2115(x64)','');
FILE_SCHEMA(('IFC2X3'));
ENDSEC;
```

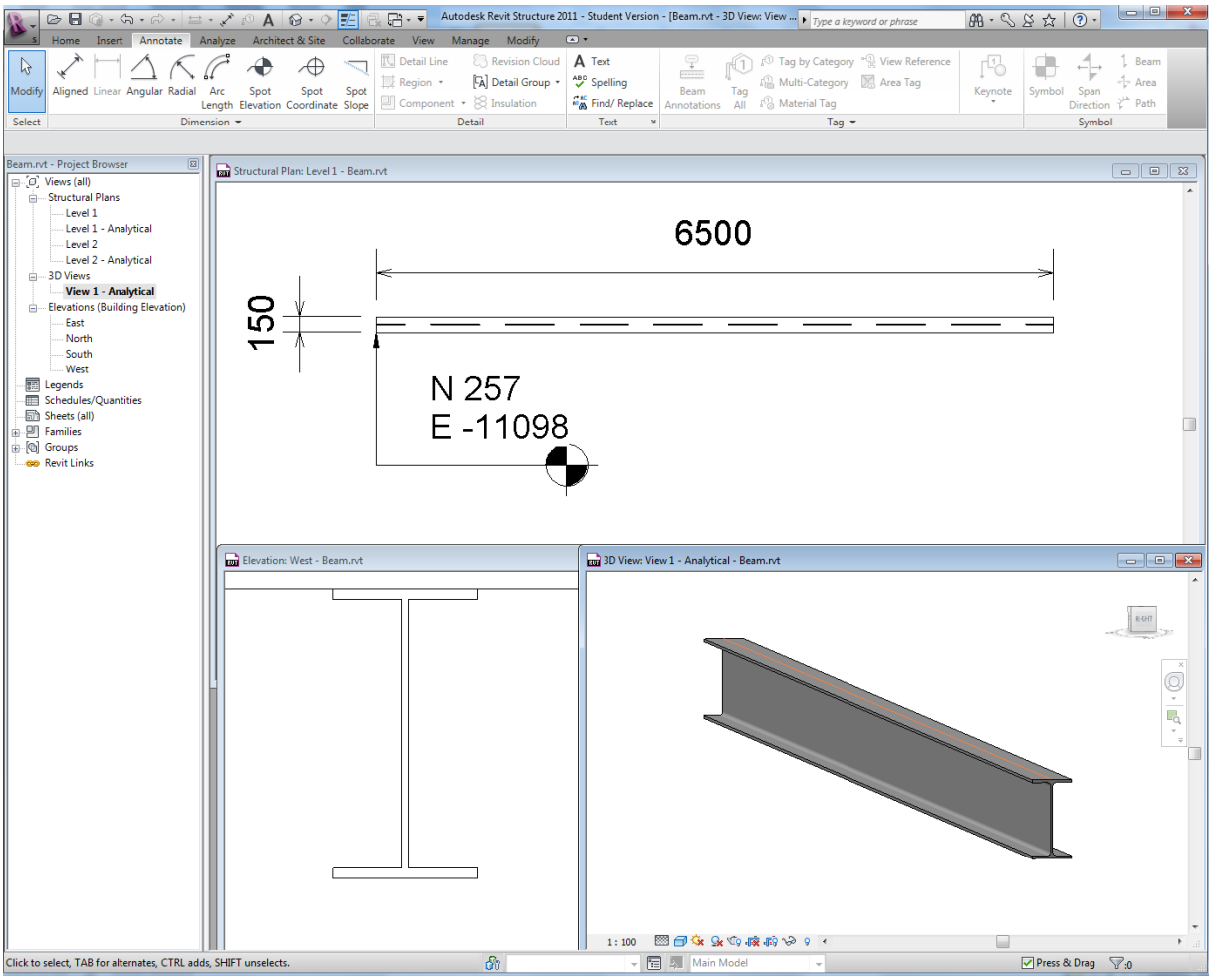
I datasektionen kodas varje enhet som en textrad och det finns endast en kolumn i dokumentet, komplexa modeller innehåller således hundratusentals textrader. Tillåtna tecken är en delmängd (tecken 32-126) av de 128 tecken som anges i det standardiserade ASCII-systemet (American Standard Code for Information Interchange), en universell kod för bokstäver och siffror som kan representeras på en datorskärm. Svenska bokstäver, exempelvis ”Åå” ”Ää” och ”Öö”, är inte tillåtna tecken i IFC-formatet och måste vid behov specialbehandlas vilket dock faller utanför ramarna för den här rapporten.

Den inbördes radordningen mellan enheterna behöver inte nödvändigtvis spegla den hierarkiska strukturen, enheterna arrangeras valfritt av den exporterande applikationens algoritmer. En enhets sub- och superenheter kan således vara utspridda både ovanför och under det aktuella elementets rad men det finns såklart fördelar i att samla ihop information i kluster ur ett granskningsmässigt perspektiv.

För att illustrera IFC-syntaxen följer här en komplett redovisning av den uppsättning enheter som används för att beskriva geometrin för en balk i vyn IFC2*3 Coordination View 2.1. I det här exemplet har en modell av en stålbalk av typen *IPE300* skapats och exporterats av ritverktyget Autodesk Revit Structure, se figur 11.

Den första enheten i exemplet, högst upp i denna hierarki, är enheten *IFCBEAM,* som delar arvlinje med de två enheterna IFCWALL och IFCSPACE vilka beskrevs i föregående avsnitt. Notera att exemplet gör avstamp på den nivå där föregående exempel avslutade, nämligen i nivån under IFCBUILDINGELEMENT.

En balk definieras i IFC-modellen som ett nära horisontellt byggnadselement som bär last mellan eller förbi stöd och vars tvärsnitt har små dimensioner i förhållande till dess längd [6].



Figur 11. Balkmodellen så som den ser ut före export till IFC-fornmat i Revit Structure.

I IFC-filen ser utskriften ut såhär:

```
#125=IFCBEAM('0Ki3Hm7wXDTQ1FyMNH2W5',#33,'IPEBeam:IPE300:IPE300:1289
30',$,'IPE-Beam:IPE300:90527',#49,#124,'128930');
```

De första siffrorna till vänster om likhetstecknet, 125, är enhetens *adress* som används av andra enheter som en referens till just denna enhet. Till höger om likhetstecknet är enhetstypen representerad med en rubrik, IFCBEAM. Parentesen omsluter de åtta kommaseparerade attributen. Informationen kan bättre åskådliggöras med en lista över attributens rubrik och tillhörande exempelvärden:

Nivå	*1*
Attribut	*Värde*
GlobalID	0Ki3Hm7wXDTQ1yFyMNH2W5 (IFCGLOBALUNIQUEID)
OwnerHistory	#33 (IFCOWNERHISTORY)
Name	IPE-Beam:IPE300:IPE300:128930 (textsträng)
Description	$ (textsträng)
ObjectType	IPE-Beam:IPE300:90527 (textsträng)
ObjectPlacement	#49 (IFCLOCALPLACEMENT)
Representation	#124 (IFCPRODUCTDEFINITIONSHAPE)
Tag	128930 (IFCIDENTIFIER)

Attributen har beskrivits i föregående exempel. Läsaren skall i detta fall uppmärksammas på att det andra, sjätte och sjunde attributet refererar till enhetsadresser nedåt i hierarkin. I det här exemplet är det fjärde attributet, IFCROOT-attributet *Name*, bokfört som ett ”$”-tecken. Detta tecken skall användas för valfria attribut som inte implementerats av den exporterande applikationen. Härnäst följer en utveckling av de adresserade subenheterna:

```
#33=IFCOWNERHISTORY(#32,#2,$,.NOCHANGE.,$,$,$,0);
```

Nivå	*2*
Attribut	*Värde*
Owning User	#32 (IFCPERSONANDORGINIZATION)
OwningApplication	#2 (IFCAPPLICATION)
State	$ (en av typerna *READWRITE, READONLY eller LOCKED*)
ChangeAction	NOCHANGE (en av typerna *NOCHANGE, MODIFIED, ADDED, eller DELETED*)
LastModifiedDate	$ (datum, heltal)
LastModifyingUser	$ (IFCPERSONANDORGINIZATION)
LastModifyingApplication	$ (IFCAPPLICATION)
CreationDate	0 (datum, heltal)

De flesta attribut i listan har en självförklarande titel. En vidareutveckling av alla subenheter blir onödigt omfattande med hänsyn till rapportens syfte, i det här exemplet är därför är förklaringen till IFCOWNERHISTORY begränsad till nivå 2.

Attributet *ObjectPlacement* refererar till en enhet av typen *IFCLOCALPLACEMENT*. Positionsangivelsen kan vara *relativ* i förhållande till ett annat objekt eller *absolut* inom det rum i vilket objektet definierats. En subenhet av IFCELEMENT skall ha en position som är absolut.

```
#49=IFCLOCALPLACEMENT($,#48);
```

Nivå	*2*
Attribut	*Värde*
PlacementRelTo	$ (IFCAXIS2PLACEMENT(2D/3D))
RelativePlacement	#48 (IFCAXIS2PLACEMENT(2D/3D))

Båda attribut refererar till ett två- eller tredimensionellt koordinatsystem, det relativa placeringsattributet är valfritt. I detta exempel refererar det obligatoriska, andra attributet till ett tredimensionellt koordinatsystem för ett rum av typen *IFCBUILDINGSTOREY*. Enheten *IFCAXIS2PLACEMENT3D* är en beskrivning av koordinatsystemets placering och axlarnas riktning inom det rummet.

```
#48=IFCAXIS2PLACEMENT3D(#47,#9,#8);
```

Nivå	*3*
Attribut	*Värde*
Location	#47 (IFCCARTESIANPOINT)
Axis	#9 (IFCDIRECTION)
RefDirection	#8 (IFCDIRECTION)

Location är en referens till origo i ett *kartesiskt* koordinatsystem, tillika utgångspunkt för objektets geometriska beskrivning i rummet. *Axis* refererar till Z-axelns riktning, *RefDirection* refererar till X-axelns riktning och i ett högerorienterat koordinatsystem är därmed också Y-axelns riktning entydigt bestämd.

#47=IFCCARTESIANPOINT((-11098.24604274508,332.0015387417489,0.));

Nivå	*4*
Attribut	*Värde*
X-coordinate	-11098.24604274508
Y-coordinate	332.0015387417489
Z-coordinate	0.

IFCCARTESIANPOINT har bara ett attribut, den uppmärksamme läsaren har kanske noterat att de till synes tre attributen i detta fall är omslutna av två parenteser. I IFC-syntaxen bokförs en lista på detta sätt, en parentes innanför den yttersta parentesen innehåller en lista med en uppräkning attribut, i detta fall de tre koordinaterna. Detta är också den första nivån i exemplet som innehåller siffror, ett språk som alla datorprogram förstår, och det är naturligtvis i den här absolut lägsta nivån som den geometriska informationen kan kommuniceras mellan olika applikationer. Lägg märke till måttsättningen i det översta fönstret i figur 11 där det står att balkhörnet befinner sig på Nordlig koordinat +257 samt Östlig koordinat -11098 relativt modellens nollpunkt. Jämför med Y=332 vilket är 257 plus halva flänsbredden, 75, vilket ger samma resultat, 332 samt X=-11098, samma siffra för båda modeller.

IFCDIRECTION definierar en riktningsvektor i två eller tre dimensioner. Förhållandet mellan X-, Y- och Z-värdena avgör vektorns riktning, ortonormerade värden rekommenderas [6].

#9=IFCDIRECTION((0.,0.,1.));

Nivå	*4*
Attribut	*Värde*
X-ratio	0.
Y- ratio	0.
Z- ratio	1.

I detta fall pekar Z-axeln i den ”ursprungliga” riktningen, axeln är inte vriden i förhållande till X- och Y-axeln. X-axelns riktningsvektor ser ut enligt nedan:

#8=IFCDIRECTION((0.,-1.,0.));

Enheten anger att X-axeln ligger i ”den ursprungliga” Y-axelns negativa riktning vilket innebär en vridning av rummet runt Z-axeln och här gäller det att vara uppmärksam för den som skall programmera en IFC-tolk. En beräkning av vektorprodukten mellan ”den nya” X-axeln och den ”urspringliga” X-axeln ger absolutvärdet av den *minsta* vinkeln mellan två vektorer men en axel kan vridas antingen positivt eller negativt med denna vinkel. Det är Y-axelns riktning som avgör vilket tecken, positivt eller negativt, som är korrekt. I detta fall skall rummet vridas 90 grader negativt runt den ursprungliga Z-axeln, se kapitel 5.4 och figur 14.

En återgång till översta nivån ger att attributet *Representation* i IFCBEAM refererar till enheten *IFCPRODUCTDEFINITIONSHAPE* som är den översta nivån i objektets geometriska beskrivning. Enheten kan som sagt innehålla mer än en geometrisk representation av samma byggnadsdel.

#124=IFCPRODUCTDEFINITIONSHAPE($,$,(#46,#123));

Nivå	*2*
Attribut	*Värde*
Name	$ (textsträng)
Description	$ (textsträng)
List of Representations	(#46,#123) (IFCSHAPEREPRESENTATION)

I exemplet är de första två, valfria, attributen utelämnade. Listan över geometriska representationer innehåller två referenser. Båda enheter är av typen *IFCSHAPEREPRESENTATION*, det första argumentet hänvisar till enheten #46 som är den översta nivån i beskrivningen av en linje som skulle kunna användas i en förenklad beräkningsmodell. Det andra argumentet, #123, är översta nivån i en detaljerad solidgeometri för användning inom exempelvis alla sorters ”visualiserande” applikationer.

#46=IFCSHAPEREPRESENTATION(#27,'Axis','Curve2D',(#45));

Nivå	*3*
Attribut	*Värde*
ContextOfItems	#27 (IFCGEOMETRICREPRESENTATIONCONTEXT)
RepresentationIdentifier	'Axis' (textsträng)
RepresentationType	'Curve2D' (textsträng)
Items	(#45) (IFCREPRESENTATIONITEM)

Det första attributet ska hänvisa till enheten *IFCGEOMETRICREPRESENTATIONCONTEXT* som definierar grundläggande egenskaper för modellen, exempelvis antal dimensioner och toleransnivå för måttangivelser. Det andra och det tredje attributet ska beskriva typ av geometri respektive sättet på vilket denna geometri framställs i IFC-modellen. Det fjärde attributet är återigen en lista som erbjuder valmöjligheten att framställa samma objekt på olika sätt, allt för att öka flexibiliteten och anpassningsförmågan till flera applikationer. I exemplet innehåller listan endast en referens, en adress till enheten #45 som är av typen *IFCPOLYLINE,* en sammanhängande kurva av linjära segment mellan ett godtyckligt antal punkter.

#45=IFCPOLYLINE((#4,#44));

Nivå	*4*
Attribut	*Värde*
List of Points	(#4,#44) (IFCCARTESIANPOINT)

Enhetens enda attribut är en lista med punkter. I detta fall beskrivs balken helt enkelt som en rät linje mellan de två punkterna #4 och #44:

#4=IFCCARTESIANPOINT((0.,0.));
#44=IFCCARTESIANPOINT((-0.,6500.));

Punktenheterna anger endast koordinater i X- och Y-led vilket förmodligen beror på att balken ritats upp i XY-planet i Revit Structure. Av informationen ovan är det också möjligt att dra slutsatsen att

balken sträcker sig från ett ”lokalt origo” till en punkt som ligger 6500 längdenheter (i detta fall millimeter) ute på Y-axeln i positiv riktning, jämför med måttsättning i figur 11.

Den andra geometriska representationen har också sitt ursprung i enheten IFCSHAPEREPRESENTATION:

#123=IFCSHAPEREPRESENTATION(#27,'Body','SweptSolid',(#122));

Till skillnad från #46 beskriver den här enhetens attribut en geometri av typen ”*Body”* som representeras av en ”*SweptSolid”*. IFC-specifikationen innehåller många olika sorters beskrivningar för hur en tredimensionell kropp kan ritas upp. *Items*-attributet hänvisar till enheten #122 som är av ett enklare slag, nämligen *IFCEXTRUDEDAREASOLID.* Denna enhet beskriver en tredimensionell kropp som *extruderas* ut till ett visst djup ur en yta vilken definieras av en sluten kurva, ungefär som en ”tredimensionell” pepparkaka trycks ut ur den ”tvådimensionella” degen med en pepparkaksform. (Eventuella hål genom en extruderad solid definieras av ett extra lager av *void elements* som är en ”tom” extrudering.)

#122=IFCEXTRUDEDAREASOLID(#119,#121,#10,6500.000000000002)

Nivå	*4*
Attribut	*Värde*
SweptArea	#119 (IFCPROFILEDEF)
Position	#121 (IFCAXIS2PLACEMENT3D)
ExtrudeDirection	#10 (IFCDIRECTION)
Depth	6500.000000000002 (positivt decimaltal)

Det första attributet refererar till ytan som skall extruderas. Det andra attributet refererar till solidens lokala koordinatsystem, vilket är underställt koordinatsystemet som definierades av enhet #48. *ExtrudeDirection*-attributet anger i vilken riktning ytan skall extruderas och det fjärde attributet anger till vilket djup detta skall göras. I det här fallet skall ytan extruderas 6500 längdenheter vilket sammanfaller med den geometriska beskrivningen i enhet #46 som anger en linje mellan två punkter åtskilda av ett avstånd lika stort som extruderingsdjupet.

Enheten *IFCARBITRARYCLOSEDPROFILEDEF* är en av fem möjliga subenheter till *IFCPROFILEDEF* och definierar en tvådimensionell, sluten kurva.

#119=IFCARBITRARYCLOSEDPROFILEDEF(.AREA.,$,#118);

Nivå	*5*
Attribut	*Värde*
ProfileType	.AREA. (en av typerna .AREA. eller .CURVE.)
ProfileName	$ (textsträng)
OuterCurve	#118 (IFCCURVE)

Det tredje attributet hänvisar till en enhet av typen *IFCCOMPOSITECURVE* (subenhet till *IFCCURVE*), vilket är en lista med delsegment av en sluten kurva. I det här exemplet är det kurvan som omsluter balkens tvärsnitt som utgör den yta som skall extruderas.

#118=IFCCOMPOSITECURVE((#53,#57,#61,#65,#70,#74,#79,#83,#87,#91,#95,#99,#104,#108,#113,#117),.F.);

Nivå	*6*
Attribut	*Värde*
Segments	(#53,#57,#61,#65…#117) (IFCCOMPOSITECURVESEGMENT)
SelfIntersect	.F. (BOOLEAN)

Det första attributet innehåller listan med alla delkurvor. Det andra attributet är endast en upplysning för den som läser textfilen och anger om det finns två eller fler delkurvor som skär varandra vilket i så fall är ett av den exporterande applikationen genererat fel som kanske inte kan hanteras av den importerande applikationen. Värdet är typen *BOOLEAN,* vilket kan vara antingen sant (.T.) eller falskt (.F.).

#53=IFCCOMPOSITECURVESEGMENT(.CONTINUOUS.,.T.,#52);

Nivå	*7*
Attribut	*Värde*
Transition	.CONTINUOUS. (en av typerna .CONTINUOUS., .DISCONTINUOUS., . CONTSAMEGRADIENT.)
SameSense	.T. (BOOLEAN)
ParentCurve	#52 (IFCCURVE)

Ett kurvsegment har tre attribut. Det första anger hur övergången mellan de två angränsande segmenten ser ut, i detta fall är det en sluten kurva vilket innebär värdet *.CONTINUOUS.*. Det andra attributet styr segmentets inbördes ordning av punkter, om värdet är *False* ska punkten med högst värde anges som den första punkten på kurvan. Det tredje attributet är en referens till den aktuella kurvan, i detta fall en *IFCPOLYLINE*:

#52=IFCPOLYLINE((#50,#51));
#50=IFCCARTESIANPOINT((-75.,-139.2999999999997));
#51=IFCCARTESIANPOINT((-75.,-150.));

En IPE-balk har ett så kallat **I**-tvärsnitt som skulle kunna ritas upp på ett enkelt sätt genom att dra 12 raka sträck eller 12 enheter av typen *IFCPOLYLINE* runt konturerna men på en mer detaljerad nivå måste de fyra, avrundade, svetsarna mellan flänsar och liv också tas med. Därför innehåller den slutna kurvan 16 och inte 12 delkurvor och förutom *IFCPOLYLINE* också kurvor av typen *IFCCIRCLE*. Dessutom måste cirklarna naturligtvis beskäras så att de passar in i den sammanhängande kurvan. Detta åstadkoms med enheten *IFCTRIMMEDCURVE* som anger ett intervall av en kurva som skall tas bort eller sparas.

#70=IFCCOMPOSITECURVESEGMENT(.CONTINUOUS.,.F.,#69);

#69=IFCTRIMMEDCURVE(#68,(IFCPARAMETERVALUE(180.)),(IFCPARAMETERVALUE(269.9999999999996)),.T.,.PARAMETER.);

Nivå	*8*
Attribut	*Värde*
BasisCurve	#68 (IFCCURVE)
Trim1	(IFCPARAMETERVALUE(180.)) (decimaltal)
Trim2	(IFCPARAMETERVALUE(269.9999999999996)) (decimaltal)

SenseAgreement	.T. (BOOLEAN)
MasterRepresentation	.PARAMETER. (en av typerna .PARAMETER., .CARTESIAN. eller .UNSEPECIFIED.)

Det första attributet refererar till kurvan som ska beskäras, i det här fallet en *IFCCIRCLE* som ska beskäras till en cirkelbåge med en båglängd motsvarande svetsens tjocklek. Det andra och det tredje attributet anger intervallet som skall *trimmas* av från kurvan där *Trim1* anger startposition och *Trim2* slutposition. I det här fallet skall kurvsegmentet bestå av en 90 graders cirkelbåge som tas ut mellan 180 och 270 grader på den definierade cirkeln:

#68=IFCCIRCLE(#67,15.);

Nivå	*9*
Attribut	*Värde*
Position	#67 (IFCAXIS2PALCEMENT2D)
Radius	15. (decimaltal)

Första attributet i cirkelenheten refererar till cirkelns placering i ett lokalt, relativt enhet #122 underordnat, tvådimensionellt koordinatsystem. Det andra attributet anger cirkelns radie, i det här fallet 15 längdenheter.

#67=IFCAXIS2PLACEMENT2D(#66,#11);
#66=IFCCARTESIANPOINT((18.55000000000005,-124.2999999999997));
#11=IFCDIRECTION((1.,0.));

Detta var en beskrivning av två olika kurvsegment och det finns naturligtvis ingen anledning att visa på fler i det här exemplet. I enheten IFCEXTRUDEAREASOLID finns det däremot två attribut som ännu inte förklarats. *Position*-attributet refererar till enhet #121 som är ett lokalt, tredimensionellt koordinatsystem. Det kan tyckas överflödigt med ännu ett koordinatsystem förutom det som angetts av *ObjectPlacement*-attributet i IFCBEAM-enheten men om en balkenhet istället modelleras som ett konglomerat av flera solider finns det en poäng med den lokala definitionen inom enhet #122.

```
#121=IFCAXIS2PLACEMENT3D(#120,#8,#5);
#120=IFCCARTESIANPOINT((0.,0.,-150.));
#8=IFCDIRECTION((0.,-1.,0.));
#5=IFCDIRECTION((1.,0.,0.));
```

(Notera att enhet #120 anger att balken, liksom i x-y-led också i z-led är placerad i tvärsnittets tyngdpunktslinje, dvs. -150 vilket är halva tvärsnittshöjden på en IPE300.)

Avslutningsvis presenteras också *ExtrudeDirection*-attributet för ordningens skull:

```
#10=IFCDIRECTION((0.,0.,-1.));
```

Detta är en, nästintill, fullständig utveckling av alla underliggande subenheter till *IFCBEAM* och som synes var detta endast den geometriska beskrivningen av objektet. I en applikation för strukturanalys är, som sagt, denna information dock endast en av flera pusselbitar i omvandlingen av CAD-modell till beräkningsmodell. I avsnitt 4.1 definierades fem grundläggande aspekter i en beräkningsmodell och frågan är nu om alla dessa kan uppfyllas av IFC-vyn *Coordination View 2.1*?

Det är med detta exempel bevisat att ett IFC-balkobjekt kan innehålla en geometrisk beskrivning som omfattar både en, för beräkningsmodeller anpassad, tvådimensionell linje, samt en, för andra applikationer bättre lämpad, detaljrik, tredimensionell solid. Indelningen av CAD-modellen i strukturelement får också anses bevisad även om exempelmodellen endast bestod av *ett* objekt. Därmed är två av fem pusselbitar på plats. Fortfarande saknas information om balkens *kopplingar*, *materialegenskaper* och *laster*. I Coordination View finns inte stöd för avdelningarna *Material Property Resource* och *Structural Load Resource* (se figur 8) som annars innehåller tre, för ändamålet, lämpliga enheter nämligen *IFCBOUNDARYCONDITION, IFCMATERIALPROPERTIES samt IFCSTRUCTURALLOAD.* Dessa enheter tillsammans med sina respektive subenheter har specificerade attribut som skulle kunna matcha informationen i en beräkningsmodell:

Enhet	*Attribut - exempel*
IFCBOUNDARYCONDITION RotationalStiffnessX/Y/Z (Node/ Edge/ Face Condition)	LinnearStiffnesX/Y/Z,
IFCMATERIALPROPERTIES	YoungModulus, ShearModulus, PoissonRatio, MassDensity, ThermalExpansionCoefficient
IFCSTRUCTURALLOAD (Single/Linear/Planar/Temperature Force)	ForceX/Y/Z, Moment X/Y/Z

I avsaknad av de här enhetstyperna är användaren av Coordination View hänvisad till utnyttjandet av *property sets.* I enheten *IFCSINGLEPROPERTYVALUE* skapas ett godtyckligt attribut. Exempelfilen innehåller över hundra enheter av den här typen. IFCPROPERTYSINGLEVALUE har fyra attribut, en rubrik, en beskrivning av attributet, ett värde och en angivelse av enhet för värdet.

Här följer några exempel på IFCPROPERTYSINGLEVALUE som tillhör balkobjektet i exempelfilen:

```
#157=IFCPROPERTYSINGLEVALUE('Volume',$,IFCVOLUMEMEASURE(33722390.000
017),$);
#176=IFCPROPERTYSINGLEVALUE('A',$,IFCAREAMEASURE(5380.999999999),$);
#170=IFCPROPERTYSINGLEVALUE('bf',$,IFCLENGTHMEASURE(150.),$);
#171=IFCPROPERTYSINGLEVALUE('d',$,IFCLENGTHMEASURE(300.),$);
#172=IFCPROPERTYSINGLEVALUE('k',$,IFCLENGTHMEASURE(25.7),$);
#173=IFCPROPERTYSINGLEVALUE('tr',$,IFCLENGTHMEASURE(15.),$);
#174=IFCPROPERTYSINGLEVALUE('tf',$,IFCLENGTHMEASURE(10.7),$);
#175=IFCPROPERTYSINGLEVALUE('tw',$,IFCLENGTHMEASURE(7.1),$);
```

De här enheterna innehåller geometriska dimensioner som kan identifieras i balktabeller så som balkens volym och tvärsnittsarea samt sex stycken tvärsnittsmått av liv och flänsar.

I exempelfilen saknas information om upplag, material och laster men i Revit Structure är det möjligt att definiera alla tre och därför också möjligt att få en IFC-fil som innehåller denna information i form av property sets.

Sammanfattningsvis är det bevisat att IFC-modellen kan bära information som är specifik både för en visualiserande solidmodell och samtidigt information som är specifik för en beräkningsmodell. Valet av vy påverkar hur informationen arrangeras i IFC-filen, *Coordination View* är anpassad för utbyte av

strukturanalytiska beräkningsmodeller men en hel del nödvändiga avdelningar för detta ändamål är utelämnade.

Det är i sammanhanget också på sin plats att påpeka att det är mjukvaruutvecklarens skyldighet att se till att IFC-strukturen implementeras på ett korrekt sätt i tolken som skall översätta den interna modellen till IFC-format och vice versa. Alla nivåer av hierarkin måste implementeras och alla relationer mellan olika objekt och egenskaper inom den interna modellen måste bokföras enligt IFC-syntaxens alla regler. Sättet på vilket detta görs blir helt avgörande för kompatibiliteten mellan avsändar- och målapplikation. IFC-formatet är bara så bra som det kan implementeras och utnyttjas av sina användare.

Certifieringstestet är utformat för att kunna bevisa IFC-tolkens funktionsduglighet men det är ändå rimligt att tro att ett sådant test inte kan simulera alla möjliga informationsdelningsscenarion. En mjukvaruutvecklare som skall programmera en IFC-parser måste vara väl insatt i IFC-strukturen så att egna fel eller fel som genererats av andra applikationer kan analyseras på rätt sätt.

5. Implementering

I föregående kapitel presenterades ett sätt att exportera byggnadsmodeller i proprietära filformat till det neutrala IFC-formatet. I det här kapitlet presenteras ett sätt på vilket dessa IFC-modeller också kan importeras och detta genom att förklara hur jag programmerat en IFC-tolk för strukturanalysverktyget BRIGADE/Plus.

5.1 Utvecklingen av en prototyp för import av IFC-modeller

Exporten av BIM-modeller är bara det första ledet i utväxlingen av data mellan två olika applikationer. För att uppnå BIM-kapacitet i ordets rätta bemärkelse krävs också en funktion som gör det möjligt att *importera* dessa modeller. För detta ändamål krävs att den importerande applikationen utrustas med en IFC-tolk som kan läsa informationen i en IFC-fil och omvandla denna till en modell i den interna modelleringsmiljön. En parser kan naturligtvis programmeras på många olika sätt men det finns förslagsvis ändå några riktlinjer att hålla sig till vid utvecklingen av en första prototyp[2]:

1. Börja med att försöka importera en enkel geometri, exempelvis en kub och en cylinder. Inledningsvis måste fokus ligga på IFC-tolkens grundläggande egenskaper så som översättning från IFC-syntaxen till applikationens egen syntax.
2. När den importerande applikationen kan översätta IFC-enheter är det möjligt att bygga upp en lämplig objektstruktur som matchar den interna byggnadsmodellen och då handlar det i första hand om att skapa alla nödvändiga *klassdefinitioner*.
3. Nästa steg är att skriva ett *skript*, instruktioner till programmet, som talar om vad som skall göras med den importerade informationen, exempelvis *rita upp geometri* eller *lägga till material och randvillkor* etc.
4. Vid det här laget kan prototypen generera en geometri i den interna byggnadsmodellen och de flesta programmeringstekniska svårigheter är lösta. Detta är en bra förutsättning för att utöka hanteringen av information från IFC-filen genom att lägga till stöd för fler IFC-enheter. En annan viktig funktion som kan läggas till i detta stadium är *relationen* mellan objekt, exempelvis, de för beräkningsmodeller, så viktiga *kopplingarna* mellan strukturelement.

Steg 1 och steg 2 kan åstadkommas på två fundamentalt olika sätt. För många programmeringsspråk, exempelvis *C++* och *JAVA*, finns en så kallad *compiler,* ett verktyg som automatiskt översätter EXPRESS-syntaxen i lämpliga klasser. Det andra alternativet är att programmera översättningen och klassdefinitionen på egen hand från grunden. Eftersom det inte finns någon tillgänglig compiler för programmeringsspråket Python har jag blivit tvungen att göra det senare. En fullständig implementering av alla IFC-enheter är förmodligen aldrig aktuellt men detta är i vilket fall som helst en arbetsintensiv process som kräver mycket programmeringstid [2]. Steg 3 och 4 innebär förmodligen mindre ”mekaniskt och monotont arbete” och mer fokus på funktionalitet och smarta lösningar inom den interna datastrukturen.

Under utvecklingen av en parser-prototyp kan det vara bra att ha en verktygslåda av användbara applikationer som ökar förståelsen för och granskningsförmågan av en IFC-modell. På internet finns en mängd sådana kostnadsfria mjukvaror. De kan delas in i tre huvudgrupper beroende på användningsområde:

- *Syntaktiska utvärderingsverktyg* – verktyg som låter användaren granska hur IFC-filen är uppbyggd rent syntaktiskt. I mitt examensarbete använde jag programmet ”IFC File Analyzer (1.2)” som genererar en Excel-arbetsbok i vilken IFC-enheterna med tillhörande attribut läggs

in i lämpliga matriser, för att göra dokumentet överskådligt läggs subenheter på separata blad. För varje enhet finns dessutom en tillhörande länk till respektive definition i IFC-dokumentationen[6] på internet.

- *Visualiserande utvärderingsverktyg* – verktyg som kan visualisera den geometriska informationen i IFC-filen. Ofta kan de här applikationerna också presentera objektstrukturen i modellen och redovisa alla individuella byggnadsdelar, med tillhörande property sets och relationer mellan enheter. Jag använde programmet ”*Constructivity Model Viewer (1.0)”*.
- *Ordbehandlare* – ett standardprogram i alla operativsystem. En IFC-fil kan öppnas och granskas i en ordbehandlare precis som vilken annan textfil som helst. Jag använde *NotePad++*.

Via BuildingSmartsAlliance:s hemsida, (http://buildingsmart-tech.org) är det möjligt att kostnadsfritt ladda ner en omfattande implementeringsguide för IFC om knappt tvåhundra sidor. På hemsidan finns dessutom hela IFC-specifikationen tillgänglig och tillsammans utgör de här två dokumenten en inledningsvis tillräckligt omfattande kunskapsbank.

Utvecklingsarbetet kräver tillgång till IFC-filer, gärna så många som möjligt! På internet kan man ladda ner kostnadsfria exempelfiler men kanske ökar förståelsen för samspelet mellan avsändarapplikation och målapplikation om det finns möjlighet att själv rita upp och exportera CAD-modeller till IFC-format i ett objektorienterat ritverktyg. Jag använde *Autodesk Revit Structure 2011*.

5.2 Exporterande applikation – Autodesk Revit Structure 2011

Autodesks programserie Revit (*-Architecture*, *-Structure* och *-MEP*) innehåller som sagt några av branschens populäraste BIM-verktyg för projektering. Konkurrerande mjukvaruleverantörer är bland andra *Bentley Systems* och finska *Tekla*. I mitt arbete har alla IFC-exporter utförts med hjälp av ritverktyget Revit Structure 2011.

Det grafiska gränssnittet påminner mycket om de vektorbaserade *AutoCAD*-programmen men Revit bygger till skillnad från de äldre ritverktygen på en objektorienterad datastruktur som tillåter användaren att skapa en parametriserad byggnadsmodell där varje byggnadselement skapas som ett objekt som kan tillskrivas objektspecifik information. I Revit skapas byggnadselement ur fördefinierade klasser, så kallade *Families,* exempelvis *Beam, Column, Plate, Wall* och *Window*. En viktig skillnad jämfört med AutoCAD-programmen är att dessa objekt är låsta inom ramarna för klassdefinitionen och inte tillåter en ”fri” modifiering av elementets geometri. Detta innebär bland annat att det inte är möjligt att radera och lägga till enskilda polygoner i polygonnätet som spänner upp den tredimensionella kroppen. Istället kan en geometri normalt endast modifieras enligt vissa objektattribut exempelvis *längd, bredd, djup* och *vinkel.* Detta sätt att rita på kräver inte samma ritskicklighet av användaren jämfört alternativet att rita fritt med linjer och kurvor, vilket kan bli ganska komplicerat i tre dimensioner men detta till priset av en reducerad flexibilitet.

I Revit Structure är det möjligt att skapa en tredimensionell konstruktionsritning och beräkningsmodell samtidigt, baserat på samma angivna data. Programmet visualiserar då två olika representationer av byggnaden på ett korrekt sätt. För arkitekt/konstruktionsändamål studeras modellen i tre olika arbetsvyer nämligen *Elevation, Section* och *3D-view* och för strukturanalysändamål finns den så kallade *Structural view*. För en beräkningsmodell är det också möjligt att lägga till materialegenskaper, upplagsvillkor och laster. Ambitionen är uppenbarligen att Revit Structure skall kunna definiera en komplett beräkningsmodell för export till strukturanalysverktyg.

Revit Structure 2011 är certifierat av Building Smart Alliance som en IFC-kompatibel applikation.

Exporten av Revit-modellen till IFC-format är fullständigt automatiserad. Användaren har möjlighet att styra valet av IFC-vy (endast Coordination View är förinställd) men det går naturligtvis inte att kontrollera exakt hur den interna geometriska representationen översätts till en IFC-geometri. Revits parser styrs av en algoritm som tar hänsyn till kroppens komplexitet. En enkel geometri, exempelvis ett rätblock översätts exempelvis till en extruderad yta med IFC-enheten *IFCEXTRUDEAREASOLID*. En mer komplex geometri, exempelvis en solid som beskriver en sammansatt ram översätts till IFC-enheten *IFCSURFACEBASEDMODEL*. I detta sammanhang är det värt att understryka att modellens detaljnivå styrs av användandet av *Family*-klasser. Om exempelvis överbyggnaden på en bromodell skapas ur en egen klass (ex. *Bridge Superstructure*) modelleras denna som en enda, sammansatt, solid, se figur 16, och oundvikligen exporteras överbygganden till IFC-format som ett enda strukturelement. Detta kan möjligtvis orsaka problem vid importen till ett strukturanalysverktyg om överbyggnaden egentligen består av flera enskilda strukturelement, exempelvis åtskilda balkar och plattor, med olika material och olika strukturdynamiska beteenden. För en god synkronisering av beräkningsmodell mellan ritverktyg och strukturanalysverktyg måste därför den strukturella uppdelningen av byggnadselement vara tydlig också för arkitekten/projektören.

5.3 Importerande applikation - BRIGADE/Plus 3.1

En målsättning för examensarbetet var att redogöra för hur BIM kan användas på ett konkret sätt tillsammans med strukturanalysverktyget *BRIGADE/Plus* som utvecklas och distribueras av Scanscot Technology.

BRIGADE/Plus är ett FEM-verktyg som riktar sig till användare inom brobyggnadsbranschen. Programmet tillåter simulering av godtyckliga strukturanalytiska problem men har också försetts med ett verktyg för de komplicerade beräkningar som uppstår i samband med analysen av trafiklasten på en brobyggnad, både statiska och dynamiska effekter kan hanteras.

Som FE-solver används en integrerad *ABAQUS*-solver, utvcklad av *SIMULIA* [11].

Den interna BRIGADE/Plus-modellen är objektorienterad i en hierarki som påminner om IFC-modellen till viss del. På den högsta nivån kan de olika objektgrupperna formuleras enligt listan nedan:

- *Parts* – alla geometriska objekt som skall analyseras är definierade som *Parts*. (Part-modulen)
- *Materials och Sections* – materialegenskaper, tvärsnitt etc. (Property-modulen)
- *Assembly* – assemblering av flera *Parts* till en sammanhängande geometri. (Assembly-modulen)
- *Loads och Boundary Conditions* – definition av laster och randvillkor. (Load-modulen)
- *Interactions* – kopplingar mellan strukturelement. (Interaction-modulen)
- *Meshes* – beräkningsmodellens *mesh*. (Mesh-modulen, en modell kan ha flera mesh:ar)

Det grafiska gränssnittet är uppdelat i tretton olika arbetsvyer, så kallade *moduler,* med specifik funktionalitet, elva för modelleringen av beräkningsmodellen i *pre-processorn*, en för *solvern* och en för visualiseringen av resultat i *post-processorn*. Med hjälp av olika funktioner i dessa moduler kan användaren enkelt formulera och simulera ett givet scenario men det finns också ett alternativ till användningen av det grafiska gränssnittet. BRIGADE/Plus tillåter nämligen också att användaren skriver och matar programmet med ett så kallat *skript*, en programkod som formulerar en godtycklig mängd order till programmet. En order till programmet att utföra en viss funktion kan således ges på två sätt, antingen genom en knapptryckning i det grafiska gränssnittet, eller genom inläsning av ett antal rader programkod från ett skript. *Skriptning* är ett mycket kraftfullt verktyg och används med fördel framför det grafiska gränssnittet vid upprepning av en lång serie kommandon. Det är således möjligt att programmera egna applikationer, så kallade *add-ons,* till BRIGADE/Plus. Mitt skript som översiktligt presenteras i nästa avsnitt innehåller ungefär 1200 rader programkod.

Python är modelleringsspråk i BRIGADE/Plus, och påstås vara ett mycket intuitivt, flexibelt och samtidigt kraftfullt programmeringsspråk som passar för utveckling av både enkla och mer avancerade applikationer (döpt efter den brittiska komedigruppen *The Monty Python's Flying Circus*) [7]. I Python är objektorienterad programmering en valfrihet vilket innebär en fördel vid skapandet av enklare applikationer. En nackdel med Python sägs vara exekveringshastigheten, det är möjligt att skriva program i andra språk som går snabbare att köra men dagens datorer är så kraftfulla att detta rimligen inte har någon betydelse, i alla fall inte för exekveringen av kortare skript.

5.4 Prototypen

Syftet med utvecklingen av prototypen har först och främst varit att konkretisera den utarbetade metodiken genom att påvisa *möjligheten* att importera IFC-modeller till BRIGADE/Plus och därför har fokus legat på steg 1-3 i den tidigare beskrivna arbetsordningen i avsnitt 4.1. Prototypen kan översiktligt delas in i tre steg med tre särskilda funktioner:

1. Inläsning av IFC-filen och kopiering av data till en Python-databas.
2. Från databasen byggs en objektstruktur upp som matchar BRIGADE/Plus-modellen.
3. Modellering i BRIGADE/Plus.

De två första stegen är med andra ord en förbehandling av IFC-informationen, omdisponeringar och beräkningar som ska trimma objekten så att de passar in i de olika BRIGADE/Plus-modulerna. Först i det tredje steget anropas BRIGADE/Plus-funktionerna som bygger upp den interna byggnadsmodellen.

Jag kommer inte att göra en detaljerad presentation av skriptets källkod mer än i de fall då detta kan anses vara värdefullt för förståelsen av programmets funktion.

Steg 1 - Inläsning och kopiering av IFC-data

I ett första steg görs en kopiering av alla IFC-enheter, rad för rad. Kopiorna bearbetas och lagras som Python-objekt i ett så kallat *dictionary* (En dictionary-databas är precis som namnet antyder, en databas med formatet av ett uppslagsverk. Varje given instans i databasen har ett ”uppslagsord”, *key,* och ett ”uppslag”, *value.*). För att knyta an till metodikkapitlet skall jag förklara skriptets funktioner genom att på samma sätt utgå ifrån balkexemplet och enheten *#125-IFCBEAM* som i detta fall skall omvandlas till en *Part* i BRIGADE/Plus.

För att förklara kopieringsprocessen av IFC-filen används för enkelhetens skull den simpla och tidigare beskrivna enheten *#45-IFCPOLYLINE*:

```
#45=IFCPOLYLINE((#4,#44));
```

Först skapas en instans i databasen, där enhetens adress, i detta fall *45,* sätts till *key* och attributen, `(#4,#44),` lagras i ett objekt på *value*-positionen. Ett sådant objekt initieras med tre attribut enligt klassdefinitionen (Python-syntax):

```
class IFC_object(object):

    def __init__(self, key, type, Arguments):

          self.key=key                  (key=45)
          self.type=type                (type=IFCPOLYLINE)
          self.Arguments=Arguments      (Arguments=(#4,#44))
```

Det första attributet är enhetens adress, det andra anger typ av IFC-enhet och det tredje innehåller enhetens attribut, där kommateringen används för uppdelning av attributen i en lista, *Arguments*. Tillsammans utgör nyckeln och objektet en instans i dictionary-databasen, *D,* enligt:

```
D = {key: value,…}
D = {45: <__main__.IFC_object object at 0x000007FFFF5BAF28>, …}
```

På samma sätt kommer också linjens refererade attribut, punkterna *#4* och *#44*, liksom alla andra IFC-enheter i textfilen att lagras i databasen enligt:

```
D = {45:<__main__.IFC_object object at 0x000007FFFF5BAF28>,
4:<__main__.IFC_object object at 0x000004FDDD6BAJ12>,
44:<__main__.IFC_object object at 0x000012HHHH5BCG32>}
```

När sista raden blivit inläst och databasen är skapad finns all information, som behövs för att köra resten av skriptet, i datorns arbetsminne och IFC-filen kan stängas eftersom den inte kommer att användas mer. Nu kan de nya Python-objektens olika attribut kommas åt enligt promt-exemplen nedan:

```
>>> D['45'].key
45
>>> D['4'].name
IFCCARTESIANPOINT
>>> D['44'].Arguments
(-0.,6500.)
```

Steg 2 - Uppbyggnad av modellens objektstruktur

Hela IFC-modellen kopieras men i det här skedet sker en utgallring av den onyttiga informationen, endast data som är kopplad till strukturanalysen skall importeras i BRIGADE/Plus-modellen.

I databasen är informationen utspridd och det finns ingen koppling mellan det refererande attributet och den refererade enheten. I detta steg kopplas de utvalda IFC-enheterna ihop så att data ackumuleras i en objektstruktur som byggs upp med en anpassning till BRIGADE/Plus-modulerna, jämför figur 1. Strukturelement blir *Part*-objekt, och material blir *Material*-objekt. I min prototyp är importen begränsad till den geometriska informationen i *Part*-modulen samt beskrivningen av ett material i *Property*-modulen. I Assembly-modulen assembleras de individuella strukturelementen till en sammanhängande byggnadsmodell. Andra viktiga parametrar så som upplagsvillkor, kopplingar och laster, som eventuellt finns i BIM-modellen, blir alltså inte importerade i denna första prototyp.

I enlighet med IFC-modellens hierarki måste informationen extraheras uppifrån och ner. I tabell 5.1 presenteras en förteckning över alla oberoende superenheter som skriptet hanterar:

Tabell 5.1 – Prototypens omfattning, hanterade IFC-superenheter.

Strukturelement	***Nivå i IFC-modellen***	***Avdelning***
IFCBEAM	*Interoperability Layer*	*Shared Building Elements*
IFCSLAB	”	“
IFCFOOTING	“	“
IFCCOLUMN	“	“
FCWALL	“	“
Relationer		
IFCRELASSOCIATESMATERIAL	*Core Layer*	*Product Extension*

IFCRELCONT.INSPA.STRUCTURE	“	“
IFCRELDEFINESBYPROPERTIES	”	”
Elevationsplan		
IFCBUILDINGSTOREY	*Core Layer*	*Product Extension*
Övriga egenskaper		
IFCPROPERTYSET	*Core Layer*	*Kernel*
Enheter		
IFCSIUNIT	*Resource Layer*	*Measure Resource*
Material		
IFCMATERIAL	*Resource Layer*	*Material Resource*

I Coordination View finns det 19 olika typer av byggnadselement definierade. Av dessa har jag valt ut 6 stycken i listan över *Strukturelement* som kan tänkas användas av IFC-tolken i Revit Structure vid exporten av en brobyggnadsmodell det vill säga *balkar*, *plattor*, *fundament*, *pelare* och *väggar*(stöd- och vingmurar).

Listan *Relationer* innehåller tre enheter, det är *IFCRELASSOCIATESMATERIAL* som kopplar ett strukturelement till ett material, *IFCRELDEFINESBYPROPERTIES* som kopplar ett strukturelement till ett *property set*, och *IFCRELCONTAINEDINSPATIALSTRUCTURE* som kopplar ett strukturelement till ett elevationsplan, se figur 10. Kopplingen till ett elevationsplan är viktig eftersom alla strukturelements lokala positioner relaterar till elevationsplanets (*IFCBUILDINGSTOREY*) globala position i IFC-modellen. Strukturelementets positionsangivelse måste således sättas i förhållande till elevationsplanets för att modellen skall assembleras på ett korrekt sätt i BRIGADE/Plus (mer om detta i steg 3).

Övriga egenskaper innehåller strukturelementens property sets. I den här prototypen används dessa enheter enbart för en gallring av IFC-objekt. Jag har programmerat en ”passkontroll” som gör det möjligt för IFC-tolken att endast släppa in byggnadselement som skall ingå i beräkningsmodellen. Alla objekt som är avsedda att ingå i en strukturanalys måste under export tilldelas en *IFCPROPERTYSINGLEVALUE*-enhet knuten till sig med ett *Name*-attribut som har värdet ”*Description:”* och ett *Value*-attribut som har värdet ”*Structural*” . På detta sätt sköts gallringen på sätt och vis redan i den exporterande applikationen vilket förhoppningsvis innebär en smidigare import av relevant data i strukturanalysverktyget.

IFC-enheter av typen *IFCUNIT* anger vilka måttenheter som använts i den exporterande applikationen, exempelvis *meter* eller *millimeter*, *grader* eller i *radianer* etc. Detta har naturligtvis betydelse för BRIGADE/Plus-användaren som skall definiera laster i överensstämmande enhet och korrekt intensitet. Angivna enheter skrivs ut som en upplysning i prompten.

IFCMATERIAL har som sagt endast två argument, *Name* och *Description* och därför kan Material-objekten i BRIGADE/Plus inte tillskrivas några fysikaliska storheter. I prototypen får strukturelementen således endast en materialbeskrivning, exempelvis ”*Concrete-C35/45*”.

Alla data som skall utgöra ett Part-objekt måste extraheras och samlas ihop på lämpligt sätt. Detta åstadkoms av flera funktioner i flera steg som bygger upp en ny objektstruktur genom att koppla attributens referenser till respektive adress i databasen ”*D*”.

Ett sätt att förklara resultatet är att presentera klassdefinitionen för ett strukturelement som ska bli ett Part-objekt:

```
class Element(object):

    def__init__(self,geo_object,origovec,refDirXvec,refDirZvec
          ,ID):

          self.geometry=geo_object      (IFC_a_closedarea, objekt)

          (#119IFCARBITRARYCLOSEDPROFILEDEF)
          self.origovec=origovec        (11098.24604274508,
                                        332.0015387417489,0.)
                                        (#47-IFCCARTESIANPOINT)
          self.refDirXvec=refDirXvec    (0.,-1.,0.) (#8-IFCDIRECTION)
          self.refDirZvec=refDirZvec    (0.,0.,1.) (#9- IFCDIRECTION)
          self.ID=ID                    (IFCBEAM128930)
```

Ett objekt av klassen ”*Element*” har fem attribut. Det första, *geo_object,* är ett *child*-objekt som innehåller den geometriska representationen. I min prototyp har jag endast implementerat stöd för solidmodellering, tillåtna IFC-geometrier är någon av IFC-typerna *IFCEXTRUDEDAREASOLID, IFCMAPPEDITEM* eller *IFCFACEBASEDSURFACEMODEL*. Precis som det finns många olika sätt på vilket en solidkropp kan definieras i IFC-modellen så finns det också många olika sätt på vilket en sådan kropp kan ritas upp i BRIGADE/Plus. Jag använder två olika ritverktyg, dels det som kallas för *solid extrude,* ett extruderingsverktyg, och dels en serie av verktyg i vilken man först ritar upp linjer, *wire*, mellan punkter som omsluter och bildar en yta, *face* (ungefär på samma sätt som en teckning ritas upp mellan numrerade punkter i en pysselbok för barn) och som tillsammans med andra, sammanhängande, ytor utgör skalet till en solid. Om extruderingsverktyget skall användas för att rita upp en solid måste *geo_object* definiera en yta. Om istället ”*face*”-verktyget skall användas måste *geo_object* innehålla alla punkter som definierar linjernas sträckning. I balkexemplet skall soliden extruderas ur tvärsnittets yta och geo_object är därför av objekttypen *IFC_a_closedarea*:

```
class IFC_a_closedarea(object):

    def__init__(self,Points,depth,extrudeDir,sub_origovec,
    sub_DirZvec,sub_DirXvec):

        self.Points=Points              (Lista med punkter runt tvärsnittet.)
                                        (#118-IFCCOMPOSITECURVE))
        self.depth=depth                (6500.000000000002)
        self.extrudeDir=extrudeDir      (0.,0.,-1.)
        self.sub_origovec=sub_origovec  (0.,0.,-150.)
        self.sub_DirZvec=sub_DirZvec    (0.,-1.,0.)
        self.sub_DirXvec=sub_DirXvec    (1.,0.,0.)
```

Både *Element*-objektet och *IFC_a_closedarea*-objektet har tre attribut som definierar respektive koordinatsystem nämligen, *origovec* som anger systemets origo, *_DirZvec* , som anger z-axelns

riktning och `_DirXvec` som anger x-axeln riktning. Denna information tillsammans med elevationsplanets (IFCBUILDINGSTOREY) position utnyttjas i *Assembly*-modulen för att translera och rotera strukturelementet till rätt position i förhållande till det globala koordinatsystemet i BRIGADE/Plus.

Det femte attributet, *ID,* i *Element*-objektet är en textsträng som används för att koppla objektet till ett elevationsplan och ett material (används också för namngivning av Part-objekten i BRIGADE/Plus).

När alla *Part*-objekt och *Material*-objekt byggts upp är steg 2 avslutat. Vid det här laget är det möjligt att rita upp en byggnadsmodell i BRIGADE/Plus med hjälp av den nyss uppbyggda objektstrukturen.

Steg 3 – Modellering i BRIGADE/Plus

Det arbete som skett innan det tredje steget är som sagt en förbehandling av informationen. Först i det tredje steget anropas funktioner i BRIGADE/Plus-modulerna. I det tredje steget genereras BRIGADE/Plus-modellen i, i sin tur, tre steg:

1. Ett objekt av typen *Element* översätts till en *Part* i *Part*-modulen.
2. I *Assembly*-modulen assembleras alla *Parts* till rätt position i det globala koordinatsystemet.
3. Material läggs till i *Property*-modulen.

I det första steget skapas en *Part* (p) enligt kommandot:

```
p = mdb.models['Model-1'] .Part(name=(str(ID)), dimensionality=THREE_D,
    type=DEFORMABLE_BODY)
```

(Punktnotationen i Python-syntaxen gör det enkelt att följa BRIGADE/Plus-modellens hierarkiska ordning. Den högsta nivån är ”*mdb*” följt av ”*models*”, följt av ”*Part*”.) En Part initieras som ett child-objekt till ”*Model-1*” med tre attribut nämligen *name* (i detta exempel: ID=`IFCBEAM128930`), *dimensionality* (tredimensionell kropp) och *type* (deformerbar kropp). I nästa steg hämtas beskrivningen av strukturelementets geometri från *Element*-attributet *geo_object*. För att extrudera solider utnyttjas först ritverktyget i *Sketch*-modulen där den extruderade ytan ritas upp på med raka linjer och cirklar på en *rityta,* eller *sketch* (s).

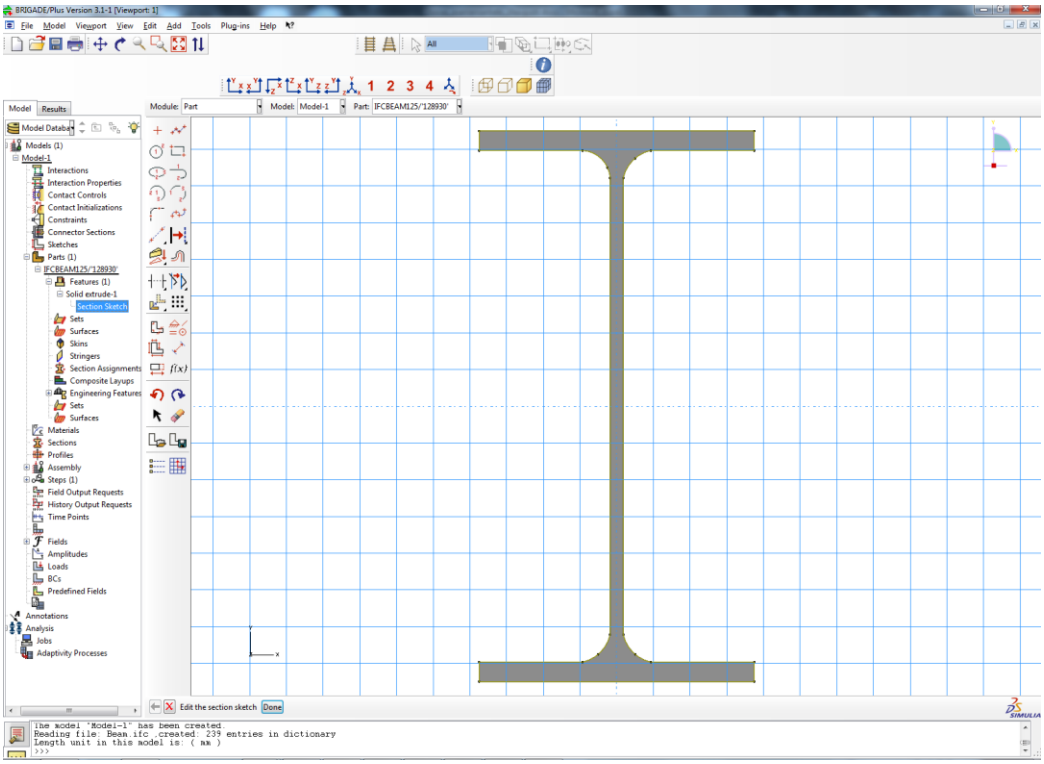
```
s.Line(point1=(Points[i].Arguments[0], Points[i].Arguments[1]), point2=(Points[i+1].Arguments[0],
Points[i+1].Arguments[1]))
```

Funktionen *Line* (som är en *metod* till *sketch*-objekt) ritar en rak linje och tar två argument, en startpunkt, *point1* och en slutpunkt, *point2.* I detta fall matas BRIGADE/Plus-funktionens argument med objektet *Points* som har två attribut, en x-koordinat, *Arguments[0],* och en y-koordinat, *Arguments[1]*. (Notera än en gång att det är dessa punktkoordinater som definierar hela geometrin i en IFC-modell.) Resultatet framgår av figur 12.

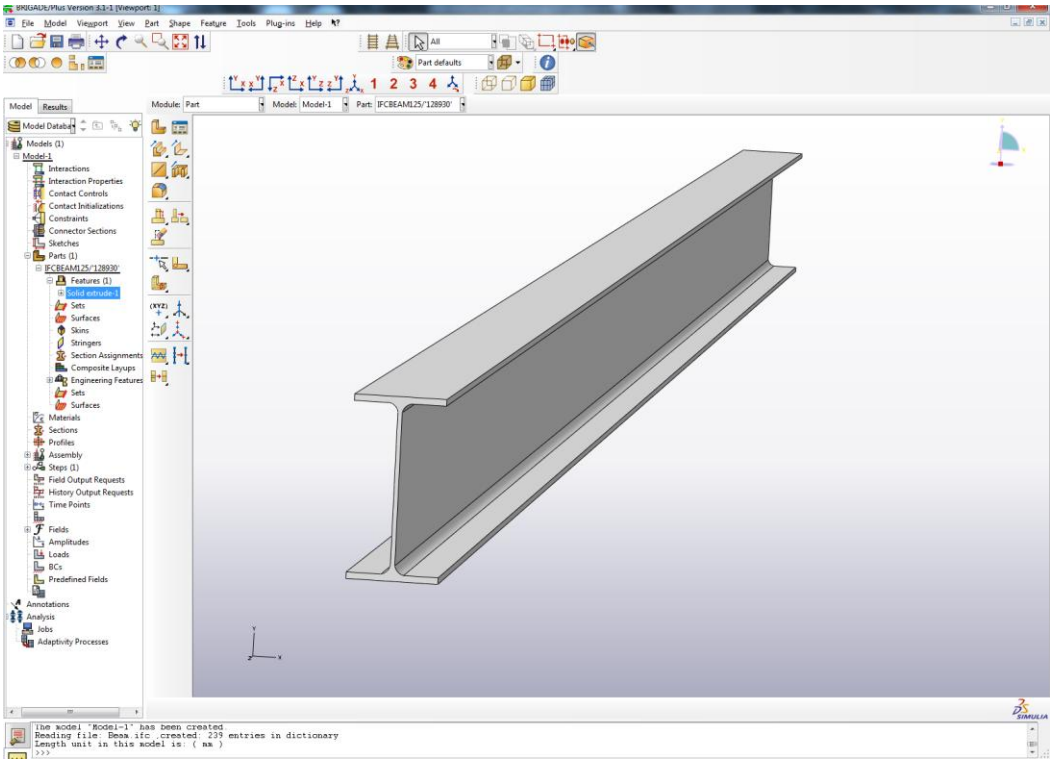
Den uppritade ytan används som underlag för extruderingsfunktionen *BaseSolidExtrude*:

```
p.BaseSolidExtrude(sketch=s, depth=float(depth))
```

Funktionen tar två argument, dels *sketch,* som i det här fallet innehåller det uppritade tvärsnittet i sketch-objektet ”s” och dels *depth* som får depth-attributet med värde ”6500.00000002”**.** Resultatet framgår av figur 13.



Figur 12. Balktvärsnittet för IFC-enheten #125IFCBEAM så som skriptet automatiskt ritar upp det i sketch-modulen i BRIGADE/Plus.



Figur 13. Balkmodell av IFC-enheten #125IFCBEAM i BRIGADE/Plus. Jämför Revit Structure-modellen, figur 11.

Solidkroppar som är av typen *IFCFACEBASEDSURFACEMODEL* skapas på ett något mer komplicerat sätt. I ett första steg används funktionen *WirePolyLine* för att skapa linjerna som

definierar ytorna. Därefter används funktionen *AddFaces* för att skapa ytorna som spänns upp av dessa linjer (se figur 16). I prototypen saknas en funktionen som omvandlar en modell av omslutande skal till en solidmodell, det är dock möjligt att åstadkomma en sådan transformation med ett av de geometriska verktygen i BRIGADE/Plus.

Assembleringen inleds med att lägga till ett *rootAssembly*-objekt (a), i vilket alla individuella byggnadsdelar, eller *Parts*, pusslas ihop till en komplett modell, i *Model-1*:

a = mdb.models['Model-1'].rootAssembly

Därefter importeras *Part*-objektet, *p*, som en *Instance* i assembleringsrummet, *a*, enligt:

a.Instance(name=str(ID), part=p, dependent=OFF)

Ett strukturelement som importeras i assembleringsrummet placeras automatiskt i origo i det globala koordinatsystemet. I det här steget skall därför strukturelementet också transleras och roteras så att det hamnar i rätt position i förhållande till de andra strukturelementen. Den globala positionen kan härledas ur sambandet mellan tre *beroende* koordinatsystem:

1. Elevationsplanets position (relativt det globala koordinatsystemet)
2. Solidkroppens position (relativt elevationsplanets koordinatsystem)
3. Den extruderade ytans position (relativt solidkroppens koordinatsystem, detta steg gäller endast för extruderade solider.)

Translationen och rotationen av ett objekt kan beskrivas som matematiska vektorproblem. Translationsproblemet är en serie enkla additioner. Exempelvis flyttas en punkt, *A*, från läget (x_1, y_1, z_1), till läget $C = (x_3, y_3, z_3)$ i ett tredimensionellt koordinatsystem genom att addera förskjutningsvektor B:s koordinater (x_2, y_2, z_2).

$$A = (x_1, y_1, z_1) \qquad (4.4.1)$$

$$B = (x_2, y_2, z_2) \qquad (4.4.2)$$

$$A + B = C = (x_1 + x_2, y_1 + y_2, z_1 + z_2) \qquad (4.4.3)$$

En summering av de tre (två för skalmodeller) *origovec*-attributen som är *förskjutningsvektorer* ger således en korrekt translation av elementet i assembleringsrummet. I balkexemplet motsvaras detta av additionen:

(IFCBUILDINGSTOREY) + (IFCBEAM) + (IFCEXTRUDEAREASOLID)

(0.,0.,0.) + (-11098., 332., 0.) + (0., 0., -150.) =
(-11098., 332., -150.) = transvec

a.translate(instanceList=(str(ID),), vector=(transvec))

Funktionen *translate* tar två argument, *instanceList* som innehåller en lista över de objekt som ska transleras och *vector* som anger förskjutningsvektorn.

Rotationen av ett strukturelement bestäms genom att räkna ut vinklarna mellan x- och z-axlarnas respektive riktningsvektorer i det ”ursprungliga” respektive ”nya” koordinatsystemet. I varje steg kan således två rotationer vara aktuella, dels en vridning runt x-axeln och dels en vridning runt z-axeln.

Vinkeln θ mellan två vektorer, exempelvis A och B, i ett tredimensionellt koordinatsystem bestäms av skalärprodukten mellan vektorerna enligt ekvation 4.4.4:

$$\theta = \arccos\left(\frac{A \cdot B}{|A||B|}\right) \qquad (4.4.4)$$

Rotationen av balkens solidmodell i förhållande till elevationsplanets koordinatsystem kan tjäna som ett tydliggörande exempel:

Sedan tidigare är det känt att *refDirXvec*-attributet i *geo_object* i det här fallet anger att den lokala x-axeln pekar i den ”globala” y-axelns negativa riktning i elevationsplanets koordinatsystem:

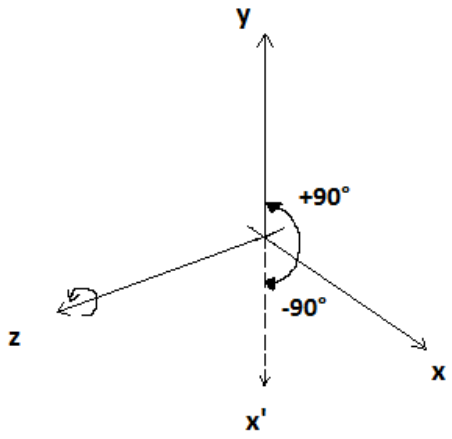
Global x-riktning `= (1.,0.,0.)`

Lokal x-riktning `= (0.,-1.,0.)`

Den minsta vinkeln mellan vektorerna kan räknas ut med ekvation 4.4.5:

$$\theta = \arccos\left(\frac{1 \cdot 0 + 0 \cdot -1 + 0 \cdot 0}{\sqrt{1^2+0^2+0^2}\sqrt{0^2+-1^2+0^2}}\right) = 90^\circ \qquad (4.4.5)$$

Soliden skall alltså vridas 90° runt den globala z-axeln , men frågan är i vilken riktning? Eftersom den ”lokala” vektorns y-koordinat har negativt tecken är det fråga om en vridning runt z-axeln i negativ riktning, detta blir tydligt genom att studera figur 14.



Figur 14. Tredimensionellt koordinatsystem. Den lokala X-axel skall vridas 90° i negativ riktning runt den globala Z-axeln.

I nästa moment skall den extruderade ytan vridas i förhållande till solidens koordinatsystem och så vidare. En Part kan således komma att roteras sex gånger med funktionen *rotate:*

a.rotate(instanceList=(str(ID),), axisPoint=(new_origo),
axisDirection=(RefDirXvec),angle=x_angle_gl)

Denna funktion tar fyra argument, det tidigare beskrivna *InstanceList*, *axisPoint* som är origo i det aktuella koordinatsystemet, *axisDirection* som anger vridaxel och *angle* som anger vridvinkel.

När alla strukturelement assemblerats återstår bara att lägga till alla specificerade byggnadsmaterial till modellen:

mdb.models['Model-1'].Material(name=str(material.name), description=str(description))

IFC-enheten *IFCMATERIAL* innehåller som sagt endast attributen *Name* och *Description,* namngivningen delas med BRIGADE/Plus-materialets attribut enligt kommandoraden ovan. I prototypen består kopplingen mellan material och strukturelement endast av en upplysning i materialets beskrivning, exempelvis:

>>> mdb.models['Model-1'].Material.description(*Concrete-C35/45,* Related Parts: „IFCCOLUMN134567‟, „IFCCOLUMN134566‟, „IFCFOOTING142300‟)

Härmed är också redovisningen av prototypens struktur avslutad. I nästa avsnitt presenteras hur prototypen testats.

5.5 Testning av prototypen

Prototypen har testats för ett mindre antal Revit Structure-exporter. Det intressantaste testet genomfördes tillsammans med en kund till *Scanscot Technology* som erbjöd en ”riktig” bromodell för utlåning till testningen. Detta är en redovisning av resultatet:

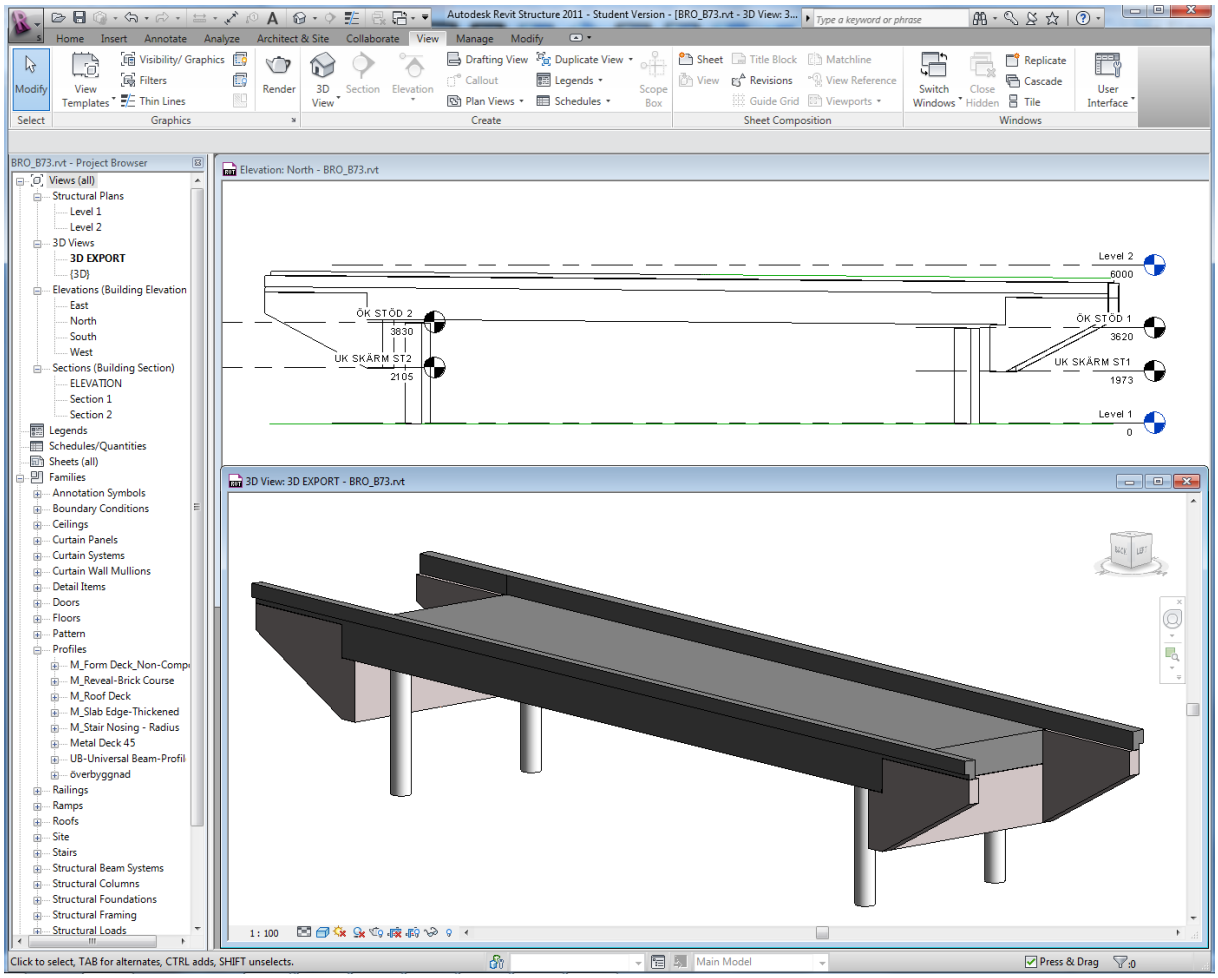
Bromodellen är skapad av broavdelningen på *Vectura* i Göteborg. Den är av ett enklare slag och i främsta hand avsedd att användas i visualiseringssyfte. Tabell 5.2 är en sammanfattning av exportens tre steg från Revit-format till BRIGADE/Plus-format.

Tabell 5.2 Export av bromodell från Revit Structure till BRIGADE/Plus

Applikation	*Filnamn*	*Filstorlek*
Revit Structure	BRO_B73.rvt	(3508 kB)
IFC-format	BRO_B73.ifc	(59 kB) (1,7% av originalstorlek!)
BRIGADE/Plus	BRO_B73.cae	(268 kB)

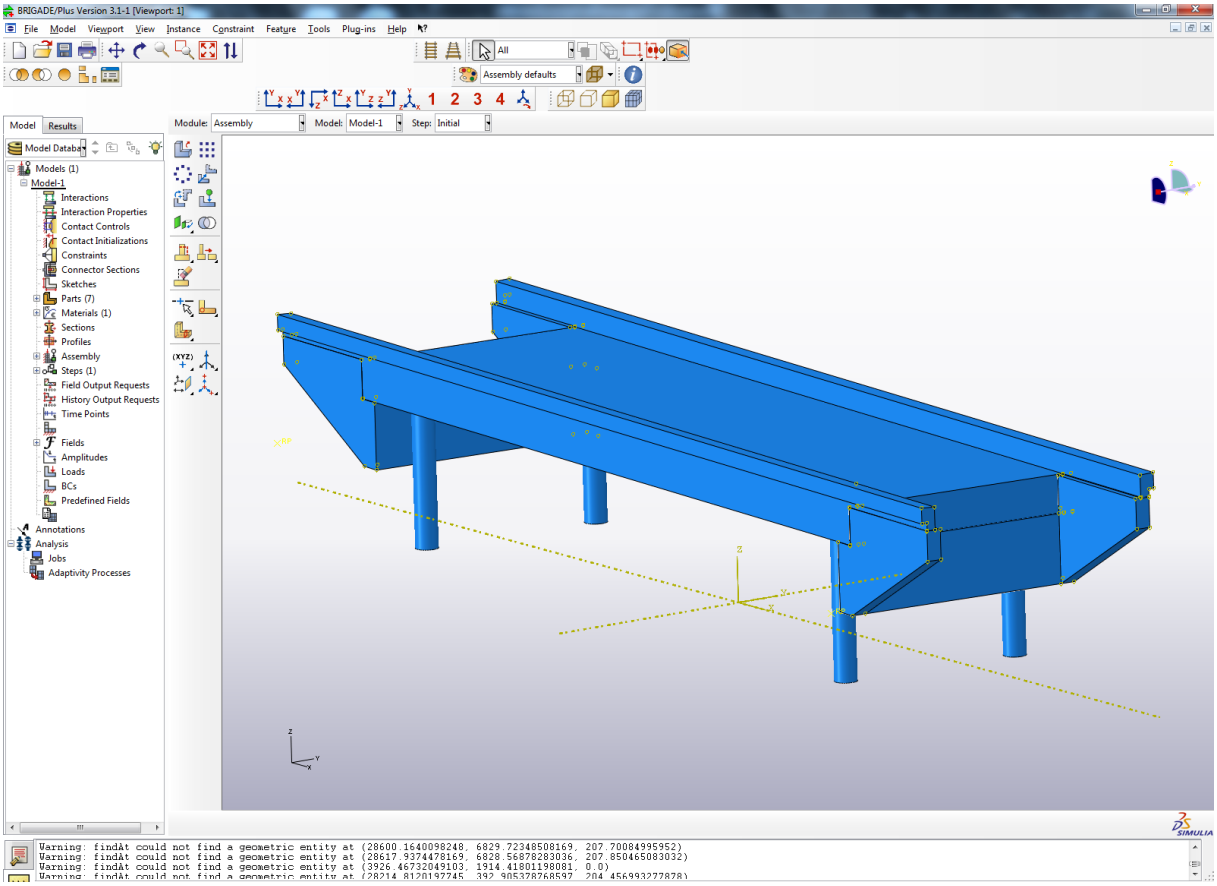
IFC-filen innehåller 67 olika typer av IFC-enheter, totalt 936 enheter. Geometrin består av 4 pelare, och tre objekt som tillsammans utgör överbygganden, dels två stödmurar och dels ett brodäck. Geometrin definieras av 145 punkter fördelade på 58 ytor. Alla strukturelement har geometrier av typen *IFCSURFACEBASEDMODEL*. Modellen innehåller två olika material, betong och stål.

Figur 15 är en skärmdump av bromodellen så som den ser ut i Revit Structure före export till IFC-format.



Figur 15. Vecturas bromodell, BRO_B73, i Revit Structure.

Figur 16 är en skärmdump av bromodellen så som den ser ut i BRIGADE/Plus efter import av IFC-filen.



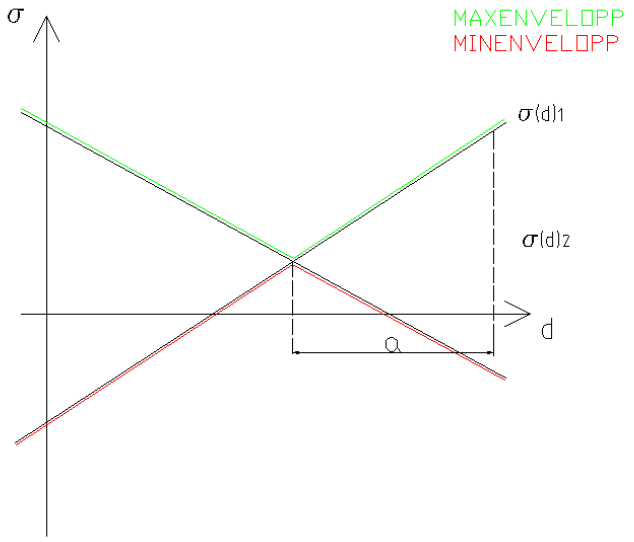
Figur 16. Bromodellen BRO_B73 som importerats via IFC-format och skapats automatiskt av mitt skript i BRIGADE/Plus.

BRIGADE/Plus-modellen är en geometriskt identisk kopia av Revit-modellen och det enda angivna materialet, betong, har importerats som en upplysande beskrivning i Property-modulen. Därmed får prototypen anses ha klarat av detta test som gick ut på att importera en IFC-geometri och IFC-material.

5.6 Problemet med solidmodellering för strukturanalysen av brokonstruktioner

I samband med utvecklingen av min prototyp blev jag uppmärksam på ett strukturanalytiskt problem vid import av solidmodeller. Om en brokonstruktion modelleras med solider i BRIGADE/Plus erhålls enligt resonemanget i kapitel 2.4, beräkningsresultat i form av *spänningar*. Ett tvärsnitt av en solid får ett mesh som består av flera element, jämför tvärsnitt i figur 6 och tvärsnitt i figur 7. Det är naturligtvis teoretiskt möjligt att manuellt integrera spänningar över dessa element för att på så sätt räkna ut snittkrafter men i praktiken finns här ett allvarligt beräkningstekniskt problem.

Beräkningsresultat redovisas normalt som *envelopperade* max- och minvärden. Dessa värden härleds ur den kombination av lastfall som ger maximalt respektive minimalt värde, enligt figur 17. Detta betyder bland annat att exempelvis ett element i tvärsnittets ovankant och ett element i tvärsnittets centrum inte nödvändigtvis antar sina respektive max- och minvärden för samma lastfall. I sammanhanget kan det påpekas att strukturanalysen av en brokonstruktion är speciell eftersom den omfattar en enorm mängd olika lastfall. Enbart analysen av *trafiklasten* genererar en uppsättning av tusentals tänkbara lastställningar för de dimensionerande *klassningsfordonen* eftersom påkänningarna som uppstår i ett givet element såklart beror av var någonstans på konstruktionen en viss last angriper.



Figur 17. Illustration av envelopperade resultat. För varje given höjdnivå, d, i tvärsnittet följer max- och min-envelopperna det högsta respektive lägsta spänningsvärdet, σ, över funktionerna σ(d)1-2.

I figur 17 representeras tvärsnittshöjden av d-axeln och spänningen för två olika lastfall av funktionerna σ(d)1-2, exempelvis två olika placeringar av en dimensionerande last som endast kan ge upphov till ett av de två lastfallen i taget. En integration över den gröna maxenveloppkurvan skulle i så fall ge maximal snittkraft precis som en integration över den röda minenveloppkurvan skulle ge minimal snittkraft. Exempelvis skulle dock en superponering över intervallet *a* att ge fel resultat eftersom dessa lastfall aldrig inträffar samtidigt. En integration av envelopperade spänningar över ett tvärsnitt är sammanfattningsvis inte praktiskt möjlig eftersom summeringen av spänningarna i tvärsnittets olika element är en summering av beräkningsresultat från olika lastfall som inte garanterat kan inträffa samtidigt. Med balk- och platt-element undviks det här problemet eftersom tvärsnittshöjden då endast representeras av ett element.

6 Slutsats och diskussion

Inledningsvis görs en rekapitulation av rapportens syfte och målsättning; syftet var att utreda och förklara hur BIM på ett konkret sätt kan effektivisera informationsutbytet mellan ritverktyg och verktyg för strukturanalys. Om möjligt skulle detta bevisas genom att programmera en prototyp för automatiserad import av BIM-modeller till BRIGADE/Plus. Rapporten skulle också på ett tekniskt sätt förklara BIM genom att behandla ämnen som objektorienterad modellering och IFC.

I kapitel 4 förklarades hur man med hjälp av det objektorienterade IFC-formatet kan exportera och importera modeller mellan olika applikationer. På frågan ”*Finns det ett sätt på vilket det går att automatisera import och omvandling av CAD-modeller till beräkningsmodeller i ett strukturanalysverktyg?”* blir svaret *Ja*. Jag vill hävda att testet av min prototyp, enligt kapitel 5.5, på ett konkret sätt visar på hur man med hjälp av IFC kan överföra CAD-geometrier från ett projekteringsverktyg till ett strukturanalysverktyg. Den importerade geometrin ritas upp automatiskt utan manuell handpåläggning vilket sparar tid åt ingenjören. Även om en import av endast *geometri* inte går att jämställa med en import av *beräkningsmodell* uppskattar jag att det i många fall är just ritarbetet som tar mest tid jämfört utsättning av laster och randvillkor vid skapandet av en ny beräkningsmodell. Detta är således en konkret effektivisering av projekteringsskedet av brokonstruktioner.

Beskrivningen av BIM-konceptet låter kanske som en utopi och det är sant att det fortfarande finns en hel del tillkortakommanden som måste lösas innan den nya tekniken får sitt definitiva genombrott. Till att börja med finns det, som jag ser det, tre grundläggande förutsättningar som måste uppfyllas för att BIM skall slå igenom på bred front, också i en så smal bransch som brobyggnadsbranschen, nämligen utveckling och implementering av nödvändig teknik, smidig och automatiserad filöverföring samt etablering av en BIM-standard för delade byggnadsmodeller.

6.1 Implementering av nödvändig teknik

Kärnan i BIM är de tekniska innovationer som lägger grunden till ett effektivare och smartare sätt att arbeta på i byggprocessen. Anammandet av denna teknik i branschen är beroende av användarvänlighet, flexibilitet och precision. I mitt arbete har jag studerat det objektorienterade projekteringsverktyget Revit Structure 2011 och förklarat hur denna applikation skiljer sig från traditionella ritverktyg, exempelvis AutoCAD. Med en objektorienterad datastruktur är det möjligt att bygga upp en byggnadsmodell som består av separata byggnadselement för vilka en godtycklig mängd information kan sparas och kommuniceras. Detta möjliggör ett disciplinöverskridande arbete i en delad byggnadsmodell som förhoppningsvis medför en effektivare arbetsprocess. Slutsatsen måste därför bli att det första steget i implementeringen av BIM utgörs av övergången från tvådimensionella vektorbaserade ritverktyg till användandet av tredimensionella, objektorienterade, projekteringsverktyg.

6.2 Automatiserad modellöverföring

Det blir kanske aldrig aktuellt att alla discipliner arbetar i en och samma applikation. En sådan utveckling är rimligen inte ens någonting att föredra om man beaktar marknadsmässiga och nationella aspekter, exempelvis ett utvecklingshämmande monopol och en osäker anpassning till nationella byggnormer. Förhoppningsvis kommer olika företag i framtiden också att arbeta med hjälp av olika mjukvaror och i så fall kommer det disciplinöverskridande samarbetet att bero på förmågan att enkelt

kunna exportera och importera byggnadsmodeller mellan olika mjukvaror, exempelvis med hjälp av IFC-formatet.

Informationsöverföring med hjälp av IFC kräver att applikationen ifråga utrustas med en tolk som gör det möjligt att importera och exportera IFC-modeller. I dagsläget finns det endast ett tjogotal IFC-certifierade applikationer. Målet för min prototyp har först och främst varit att bevisa *möjligheten* att importera IFC-modeller till BRIGADE/Plus. För att uppnå ”kommersiell funktionalitet” krävs naturligtvis en mycket större arbetsinsats. I detta sammanhang vill jag också passa på att nämna att jag har läst civilingenjörsutbildningen i *Väg och vattenbyggnad* med inriktning mot *strukturmekanik* och att jag vid examensarbetets början aldrig tidigare programmerat i Python eller något annat programmeringsspråk. Mina programmeringstekniska erfarenheter vid denna tidpunkt var endast de som alla ingenjörsstudenter på LTH får i och med de obligatoriska momenten av skriptning i *MatLab*. Därför kan upplägget på mitt skript förmodligen diskuteras och kritiseras av någon med större programmeringskunskaper. Mina kvalifikationer för det här uppdraget ligger kanske snarare i min förmåga att utvärdera IFC-modellen i avseendet att använda som underlag för en strukturanalys och på den punkten tycker jag att resultatet trots allt visar på potentialen i IFC-formatet, om än med vissa brister. I kapitel 5.5 finns ett exempel på hur en IFC-modell importeras till BRIGADE/Plus som en exakt geometriskt kopia av Revit Structure-modellen men jag måste samtidigt understryka att det inte är en beräkningsmodell som importeras för det finns fortfarande en hel del bitar som saknas exempelvis upplagsvillkor och laster.

En informationsdelning ställer som sagt höga krav på tekniken och applikationens förmåga att skriva ut och tolka in data på rätt sätt och det är svårt att veta till vilken grad informationsdelningen kan komma att utvidgas. Det finns, så som beskrivet i kapitel 4.1, ett informationsglapp mellan den traditionella konstruktionsritningen och beräkningsmodellen som inte bör underskattas. Även mellan ett ritverktyg och ett strukturanalysverktyg som båda två är objektorienterade kan det uppstå problem. I samband med utvecklingen av min prototyp blev jag uppmärksam på ett strukturanalytiskt problem vid import av solidmodeller. En integration av envelopperade spänningar över ett tvärsnitt är omöjlig vilket leder till den frestande slutsatsen att importerade geometrierna måste vara av typen *linjer* eller *skal* som kan modelleras med *balk-* och *platteori.* I avsnitt 3.4 finns ett exempel som beskriver hur en balks geometri kan definieras på två olika sätt i IFC-syntaxen, dels som en *linje* och dels som en *solid* vilket kan utnyttjas för att kringgå det här problemet. Det finns emellertid även en alternativ och kanske något mer tilltalande lösning på samma problem. I ett projekt som bedrivits parallellt med det här projektet har man lyckats finna ett sätt att beräknas snittkrafter i solidmodeller genom att först integrera spänningarna i tvärsnittet innan dessa envelopperas.

Avslutningsvis, IFC-formatet är bara så bra som det kan implementeras och utnyttjas av sina användare. Det är mjukvaruutvecklarens skyldighet att se till att IFC-syntaxen implementeras på ett korrekt sätt i tolken som skall översätta den interna modellen till IFC-format och vice versa. Det är möjligt att önska sig en exakt export och import av byggnadsmodeller mellan olika applikationer som inte kräver någon som helst handpåläggning men vi bör förhålla oss ödmjuka inför svårigheterna i denna problemlösning. Olika discipliner har olika kompetenser men också olika syften i sitt användande av modeller, därför är det också rimligt att tro att det trots allt kommer att krävas en mänsklig hjärna som tolkar den importerade modellen och gör vissa kvalificerade korrektioner för att hela omvandlingsprocessen från ritning till beräkningsmodell skall bli korrekt. Det viktigaste är att BIM-processen underlättar samarbetet mellan människor och eliminerar så mycket dubbelarbete som möjligt.

6.3 Branschstandard för BIM-modeller

Så länge det inte finns ett standardformat för BIM-modeller finns det kanske också en osäkerhet bland mjukvaruleverantörer i branschen kring utvecklingsarbetets inriktning. Etableringen av ett standardformat, exempelvis IFC-modellen, ökar säkerheten för investeringar i form av exempelvis arbete mot en IFC-certifiering. IFC-modellen bygger på en öppen specifikation som publiceras och uppdateras på internet av Building Smart Alliance. Det neutrala filformatet kan implementeras av alla mjukvaruutvecklare i världen vilket i så fall ger en byggnadsmodell som är helt oberoende av mål- eller mottagarapplikation.

I kapitel 4 har jag redogjort för IFC-modellens ambitiösa bredd och förmåga att verka som en disciplinöverskridande och gemensam byggnadsmodell. Jag har studerat IFC-vyn IFC2x3 Coordination View vilken matchar BRIGADE/Plus objektstruktur någorlunda bra men det finns flera avdelningar med koppling till strukturanalys som inte omfattas av specifikationen vilket leder till tankar om alternativa vyer. Det är av praktiska skäl ogörligt att programmera en IFC-tolk som stöder hela IFC-specifikationen och kanske är det så att IFC-modellens framtid hänger på hur bra BuildingSmartAlliance kan anpassa och matcha valet av standard-vyer till branschens olika applikationer.

6.4 BIM – fördelar och nackdelar

Det finns all anledning att tro att alla nödvändiga förutsättningar kan uppfyllas på sikt. Sammanfattningsvis, att arbeta med BIM har många uppenbara fördelar jämfört en konventionell arbetsprocess:

- *Bättre visualisering av byggnadsmodellen* – tredimensionella modeller ökar förståelsen för konstruktionen och underlättar tredimensionella analyser exempelvis kollisionskontroller.
- *Utökat samarbete*– En gemensam byggnadsmodell tillåter att flera människor arbetar tillsammans samtidigt och en mera lättillgänglig information sparar mycket tid om nödvändiga korrigeringar kan göras på ett tidigt stadium under projekteringen.
- *Ökad produktivitet* – en objektorienterad byggnadsmodell tillåter automatisering av tidsödande aktiviteter så som *mängdning*, *kollisionskontroller, arbetsscheman* och *budgetering.*
- *Iterativt arbetssätt* – ett smidigare informationsflöde mellan de olika disciplinerna gör att flera alternativa förslag hinner testas under en given projektering.
- *Snabbare leverans* – disciplinöverskridande arbete förkortar möjligtvis projektets leveranstid.
- *Reducerade kostnader* – tid är pengar, BIM sparar tid.

Eftersom den ackumulerade erfarenheten av arbetet med BIM-modeller i branschen fortfarande är relativt liten är det kanske svårt att veta huruvida alla dessa fördelar också kan infrias i verkligheten. Lika svårt är det naturligtvis att säga någonting om nackdelarna med att arbeta med BIM men det är ändå möjligt att identifiera vissa kritiska ämnen:

- *Teknisk oförmåga* – En BIM-applikation exempelvis Revit Structure kan i förhållande till AutoCAD upplevas som något mer begränsad och mindre flexibel i de fall projektören skall rita en typ av konstruktion som inte låter sig modelleras med hjälp av de fördefinierade klasserna i programmet. BIM-applikationer måste vara lika bra eller bättre än de verktyg de ersätter. Användaren måste kunna lita på att applikationen gör rätt med god precision.

- *Kompatibilitet* - BIM medför höga kompatibilitetskrav, filöverföringen mellan aktuella applikationer måste vara smidig och exakt. I annat fall riskeras användarens tilltro till tekniken.
- *Det juridiska problemet* – vid arbetet med en delad BIM-modell kan det möjligtvis uppstå konflikter kring ”vem som gjort vad i modellen” och vem som *äger* informationen. Ritningar och beräkningsmodeller är mycket dyrbara ägodelar om man exempelvis likställer värdet av antalet arbetstimmar med den konstruerade modellens värde.

Det är inte otänkbart att en majoritet av byggbranschens alla företag inom en ganska snar framtid kommer att övergå till att arbeta uteslutande med BIM-teknik. Drivkraften bakom förändringen är helt avgörande som jag ser det, det handlar om att effektivisera. Naturligtvis tar mjukvaruleverantörerna betalt för den nya tekniken men en ökad produktivitet hos BIM-användaren motiverar också en högre kostnad.

I ett större perspektiv är det möjligtvis samhället i stort som är den största vinnaren vid övergången till BIM inom byggbranschen om fler noggrant utformade, hållbara och miljövänliga byggnader snabbare kan projekteras och byggas samt därefter underhållas effektivare. Kanske har nu byggbranschen påbörjat en mycket viktig resa som kommer att förändra inte bara arbetsuppgifterna utan också arbetsrelationerna mellan olika discipliner på ett betydelsefullt sätt.

7 Referenser

[1] Ottosen Saabye, N. & Petersson, H. *Introduction to the Finite Element Method*, Lund: Prentice Hall, 1992

[2] Loffredo, D. *Fundamentals of STEP Implementation*, Troy, NY, STEP Tools Inc, 2003

[3] Isaksson, T. & Mårtensson, A. *Byggkonstruktion*, Lund: Studentlitteratur, 2005.

[4] Khemlani, L. *The IFC Building Model: A Look Under the Hood*, (2004), april 2011 [Internet] http://www.aecbytes.com/feature/2004/IFCmodel.html

[5] Building Smart Alliance, *Industry Foundation Classes, Basic Information*, (2011), juli 2011 [Internet] http://www.ifcwiki.org/index.php/Basic_Informations

[6] Building Smart Alliance *IFC 2x3 Technical Corrigendum*, (2011), juli 2011 [Internet] http://buildingsmart-tech.org/ifc/IFC2x3/TC1/html/index.htm

[7] Dawson, M. *Python Programming For The Absolute Beginner 3rd ed.*, Boston, MA: Course Technology, 2010

[8] Laiserin, J. *Comparing Pommes and Naranjas*, (2002), juni 2011 [Internet] http://www.laiserin.com/features/issue15/feature01.php

[9] Eastman, C. **& Sacks, R., & Liston, K.** *BIM Handbook – A Guide To Building Information Modeling for Owners, Managers, Designers, Engineers and Contractors*, Hoboken, NJ: John Wiley & Sons Inc., 2010

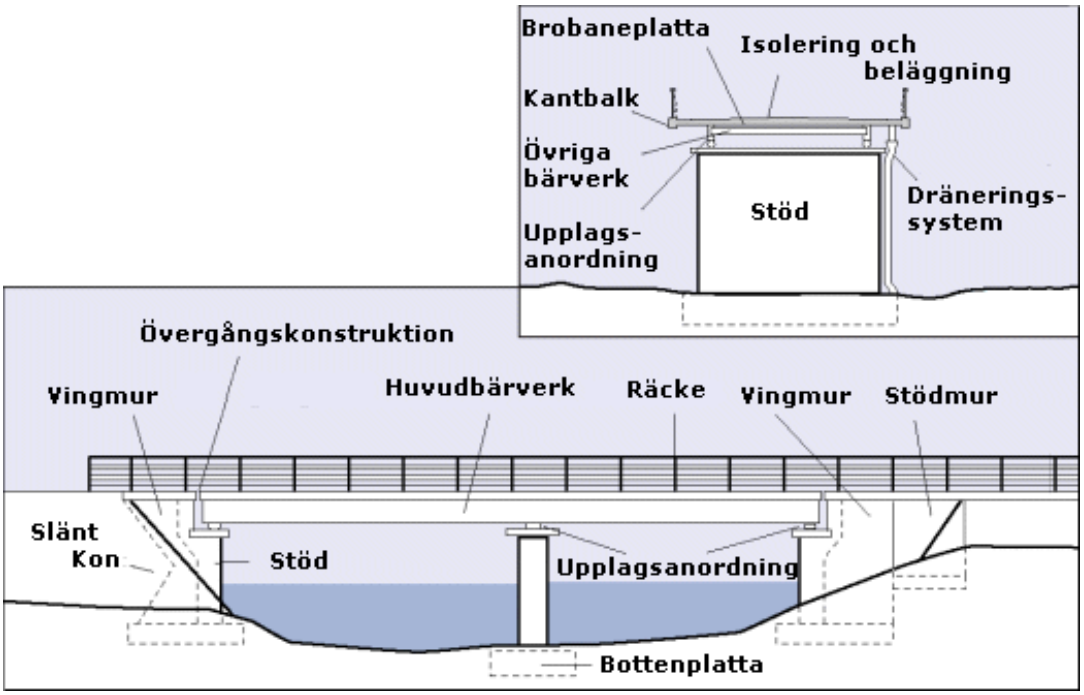
[10] Building Smart Alliance *Certification*, (2011), juli 2011 [Internet] http://buildingsmart-tech.org/certification/ifc-certification-2.0/ifc2x3-cv-v2.0-certification/participants

[11] Scanscot Technology *BRIGADE/Plus - Introductory Course Material*

8 Bilagor

8.1 Brokonstruktionens geometri

En brokonstruktion delas normalt in i två delar, en *överbyggnad* och en *underbyggnad* (för längre spännvidder kompletteras överbygganden med *pyloner* och *upphängning*). Överbyggnaden omfattar *brobaneplattan*, som brygger vägen över ett hinder samt alla konstruktionsdelar som medger en förmedling av trafiklasterna till de underliggande stöden i underbyggnaden. Exempel på huvudbärverk i överbygganden som skall bära i brons längdriktning är plattor, balkar, fackverk, hängverk, spännverk eller bågar. Sekundärbärverk, som skall bära i brons tvärriktning består av exempelvis balkar, fackverk eller stänger. Underbyggnaden tar upp laster från överbyggnaden och för ner dessa till brons fundament. Exempel på bärande element i underbygganden är pelare, ving- och stödmurar. En mer omfattande bromodell inkluderar även grundläggning som består av brons fundament, exempelvis kassuner, bottenplattor, plintar och pålar.



Figur 18. Brokonstruktionens olika bärverk.

8.2 Lista över IFC-certifierade mjukvaror

Listan är hämtad från [10] och gäller t.o.m. juli 2011:

Participants of the official buildingSMART
IFC2x3 Coordination View V2.0 certification process

Software Developer	*Software Application*	*Exchange Requirement*	*Export/Import*	*Status*
Archimen	Active3D	- (*)	Import	in progress
Autodesk	AutoCAD Architecture	Architecture	Import & Export	in progress
Autodesk	AutoCAD MEP	BuildingServices	Export	in progress
Autodesk	Revit Architecture	Architecture	Import & Export	in progress
Bentley Systems	Bentley Architecture	Architecture	Import & Export	in progress
Cad-Quality	CADiE Sähäkkä	BuildingService	Import	in progress
Data Design System	DDS-CAD MEP	BuildingService	Export	in progress
Design Data	SDS/2	Structural	Import & Export	in progress
Gehry Technologies	Digital Project	Architecture	Import & Export	in progress
Graphisoft	ArchiCAD	Architecture	Import & Export	in progress
International Training Institute ITI	Benchmark	BuildingService	Export	in progress
NEMETSCHEK Allplan	Allplan	Architecture	Import & Export	in progress
NEMETSCHEK North America	Vectorworks	Architecture	Import & Export	in progress
NEMETSCHEK SCIA	Scia Engineer	Structural	Import & Export	in progress
Plancal	nova	BuildingService	Import & Export	in progress
Progman	MagiCad	BuildingService	Export	in progress
Solibri	Solibri Model Checker	- (*)	Import	in progress

Tekla	Tekla Structures	Structural	Import & Export	in progress
VIZELIA	Facility on line	- (*)	Import	in progress

(*) Exchange Requirement applicable only to export, for import all three exchange requirements have to be supported.