IFC Dataflow

An open framework for connecting BIM web services

Marijn Laurens van Aerle
Maart 2015

Eindhoven University of Technology
Department of the Build Environment
Design Systems Group

Graduation committee:
Prof. dr. ir. Bauke de Vries
Dr. Dipl.-Ing. Jakob Beetz
Ir. Thomas Krijnen
1. Abstract

Construction projects are often one-of-a-kind projects with respect to the design of the building as well as to the organization of the project, the building site and to the processes used. In this research a novel framework is proposed that allows creating formal, graphical and flexible representations of processes that can be communicated with other stakeholders and that can be re-used across projects. Additional tools are created to facilitate connecting different BIM web services when using these workflows so that the workflows can be automatically executed with immediate graphical feedback. The framework presents a significant improvement over the current manual approach for these processes and requires a fraction of the effort required for creating a custom application for a specific process. A prototype implementation is developed and used to validate the framework on an example scenario.
2. Table of Contents

1. Abstract
2. Table of Contents
3. Introduction
4. Problem statement
   4.1. Key Characteristics
      4.1.1. Usability
      4.1.2. Extensibility & Adaptability
      4.1.3. Open
4.2. Related work
   5.1. Introduction
   Point solutions
   Query language approaches
   Toolkits and libraries
   5.1.4. BIM Service interface exchange (BIMSie)
5.2. Summary of reviewed literature
6. Dataflow programming
   6.1. Introduction
   6.2. Examples of commercial dataflow environments
   6.3. Dataflow programming in the AEC domain
      6.3.1. Grasshopper
      6.3.2. Dynamo
      6.3.3. Conclusion
7. Framework overview
   7.1. Overview
      7.1.1. Graph architecture
      7.1.2. Node specification
      7.1.3. Bimsie types
      7.1.4. Data transfer
      7.1.5. Servers & distribution of workload
      7.1.6. Existing services / wrapper
      7.1.7. BIM Service API
7.2. Open source projects
   7.2.1. Flood
   7.2.2. BIMserver
7.2.3. BimQL  
7.2.4. mvdXMLChecker  
7.3. Framework Architecture  
    7.3.1. Implementation notes  
8. Dataflow nodes  
    8.1. Introduction  
    8.2. Generic nodes  
        8.2.1. Input string  
        8.2.2. Input number  
        8.2.3. Show output  
        8.2.4. Eval  
        8.2.5. Add  
        8.2.6. Multiply  
        8.2.7. Subtract  
    8.3. Bimsie nodes  
        8.3.1. Get bimserver  
        8.3.2. Get project  
        8.3.3. Get revision  
        8.3.4. Merge models  
        8.3.5. Get object info  
    8.4. Extension nodes  
        8.4.1. BimQL query  
        8.4.2. mvdXML  
9. Example scenario  
    9.1. Introduction  
    9.2. Framework implementation  
10. Performance  
    10.1. Introduction  
    10.2. Network timings and data transfer  
    10.3. Performance Improvements  
11. Conclusion and future work  
    11.1. Conclusion  
    11.2. Future work
3. Introduction

Construction projects are often one-of-a-kind projects with respect to the design of the building as well as to the organization of the project, the building site and to the processes used. They also often have multiple stakeholders that are not always aligned, which has traditionally hampered the establishment of integrated information systems that allow easy access to distributed information and responsibility management.

As part of an ongoing process to improve itself there has been a trend in the building industry to increase the efficiency of information distribution among stakeholders by using digital tools. This trend started with moving from traditional drawings to 2D CAD, then evolved into 3D modeling, currently the state of the art is BIM (Building Information Modeling). BIM is defined by the Building Smart Alliance as “A digital representation of physical and functional characteristics of a facility... and a shared knowledge resource for information about a facility forming a reliable basis for decisions during its life-cycle; defined as existing from earliest conception to demolition.”

BIM is increasingly considered as a method to enrich building models with data, create better virtual prototypes and improve the building design & construction process in general. Typically BIM is used in a process where participants require distributed access to the model. The scope of BIM is expanding from collaboration within companies and design teams to multi disciplinary collaboration through the use of a BIMserver. “A BIMserver is a collaboration platform that maintains a repository of the building data, and allows native applications to import and export files from the database for viewing, checking, updating and modifying the data.”

There are several solutions available to improve BIM collaboration, ranging from open source tools like the Open Source BIMServer, more industrial and closed solutions like the EDM Model Server or Onuma to tools more geared towards specific CAD

---

1 "About the National BIM Standard-United States | National ..." 2013. 3 Mar. 2015 <http://www.nationalbimstandard.org/about.php>
applications like Revit Server\(^7\), Tekla BIMsight\(^8\) or the Graphisoft BIMserver\(^9\).

A fundamental premise of this research is that project collaboration in the construction industry cannot be done via a single project model, server or database. Different stakeholders will continue to use different engineering, design and management models and endpoint solutions tailored for their specific application domain in order to remain the most efficient, flexible and innovative. The emphasis therefore lies on connecting the different tools together and we propose a new approach for connecting different server based tools together in way that is highly configurable by the building professional. The goal is that this will lead to improved efficiency in a distributed asynchronous multi-model-based collaboration process.\(^{10}\)

When building professionals implement server based solutions new challenges are surfacing that need to be addressed. The biggest of these challenges is interoperability. Interoperability can be divided into two main areas, connecting formats and connecting services. Connecting formats means that different CAD packages have limited support for importing and exporting their data, often resulting in data loss. Open source standards like IFC, IFD and many more can help with this. Connecting services is a fairly new challenge resulting from the recent rise in server based approaches.

Much like the different endpoint tools used by building professionals there are also different server based tools becoming available to work with building models. Stakeholders want to maintain control over their data and services, but for an efficient construction and design process communication is a necessity. It is therefore becoming more relevant to find ways to connect these services in a maintainable way.

When looking at connecting data and services in the building industry we will focus most on forming intercompany connections. Different companies often have the most trouble when exchanging data and communicating. Intercompany connections can be

---


made on different levels. At the lowest level people need to be able to communicate efficiently. The second step is that the endpoint solutions need to be able to read in the different data formats that they can import/export, allowing model data to be transferred. When this exchange works the issue becomes how to efficiently exchange building models themselves, model servers are a solution for this problem. Finally when the exchange using model servers works the issue becomes how to connect the different model servers and the data they hold. This research has this interaction between model servers as the primary topic.

At first the problem statement will be covered, including the key characteristics usability, extensibility and open standards. Then related work in research and existing tools available will be reviewed. Then dataflow programming will be introduced as a method of modelling programs and interactions after which the created prototype will be discussed with regards to the implementation method and the different dataflow nodes that have been created. An example scenario has been created that demonstrates the approach. Finally we will review some considerations about the performance of the system and reach a conclusion and future work recommendation.
4. Problem statement

Because of the increased use of BIM, model servers and web services within the AEC industry more building data is generated than ever before. This data is typically created within information silos in separate tools, services and servers. Even when services are accessible through the web they are often accessed manually or by custom applications. By presenting a novel framework for creating formal, graphical and reusable representations of the processes that require the use of BIM services we propose that communication and efficiency is significantly improved by removing the need for manual steps and providing a method to communicate the workflow to other building professionals.

The framework provides a method for describing workflows as dataflow graphs that can be executed and provide a concrete implementation of the operations that need to be performed, so that they can yield concrete output data in the form of building models or textual data. Workflows can be composed to create higher level workflows.

Additionally building professionals often want to work with building data interactively as underlying information may change over time and different scenarios may need to be tested. Furthermore to get relevant results from the data often multiple sources have to be combined. Another challenge is that building projects differ greatly in stakeholders, complexity and character, often requiring custom made solutions for a specific project.

To help building professionals solve this pressing need for relevant information new approaches are required. There is a need for both flexibility and composability, which means that the tool should be flexible enough to be relevant in many different projects, and that the tools should be composable so that knowledge can be shared across projects and companies. However the usability of the given tool should match the skill level of building professionals, so that it is accessible to them and not only to trained programmers.

Provided that online model servers are used to store model data the next question then is how to efficiently access this data. Not only accessing the complete building model but also sub-parts of it, or only the metadata of a building model, making sure that the most recent information is used. Obtaining this information in time is critical for an
efficient building design process.

Most model servers provide APIs (Application Programming Interface) or sometimes a GUI (Graphical User Interface) to access the model data on the server. However usually there is not only the model server itself involved but also input from and output to a wide array of different (web based) services. For example, clash detection, validation, viewers, simulation services, cloud based renderers, building sensors or ERP / Finance / Asset management services. All these different services can play a role in the building process so it is important to have a service platform where these different services can be integrated (See fig.). Our approach is to do this via REST web services, as is demonstrated in the framework overview.

Fig. Service layer

Several technical challenges arise when creating this service layer:

Data needs to be transferred efficiently between the different services. This data might contain model data that includes geometry and therefore might be very large.
Compression techniques are important in this regard. Furthermore a common format is important to transfer the data in, different standards exist for exchanging data. The data will be transferred over the web, and therefore moves over HTTP.

Another challenge is the fact that existing services might not provide for the necessary transport mechanisms, for example a HTTP interface, or they might not be fit for use as a server at all. In these scenarios wrappers will need to be created so that the existing service can be exposed over HTTP.

Finally the coordinating and distributing the workload among the different services can be challenging. Usually there is a specific ordering in which the different tasks need to be executed, and certain tasks can depend on the output of previous tasks to be able to do their work. Therefore it is very important to provide an efficient way to specify how the work is coordinated. In this coordination the fact that some work can be done in parallel can potentially be exploited for performance gains, which leads to interesting challenges around concurrency.

An example of these challenges is when data from model server A needs to be queried and then validated, then data from model server B also needs to be queried and merged with the data from server A. In this scenario both the servers can start doing the query simultaneously, however only after both are done the merge operation can be started.

### 4.1. Key Characteristics

Currently several tools and methods exist to extract data from building models. Either in an ad-hoc fashion or through web services. However we propose a better solution that has a combination of several characteristics that existing approaches do not have:

- **Usability**: The skill level required to use the tool should match with the skill level of building industry professionals, not computer programmers.
- **Extensibility & Adaptability**: The project should be extensible by building industry professionals because building projects vary hugely in scope and complexity and often require tailored solutions to provide the required information.
- **Open**: The tool should be vendor independent, to stimulate innovation across the industry. It should work with building models using open standards such as IFC.
4.1.1. Usability

The definition of usability can be found in ISO 9241-11\footnote{ISO, WD. “9241-11. Ergonomic requirements for office work with visual display terminals (VDTs).” The international organization for standardization (1998).}: “The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use”

In this context effectiveness is the ability of users to complete tasks using the system, and the quality of output of those tasks, efficiency is the level of resource consumed in performing the tasks and satisfaction is the users subjective reaction to using the system.

To achieve good usability it is first important to designate the group of users that the product targets. In our case this group is building professionals in the broad sense. That means among other groups: Architects, Engineers, Contractors, Construction workers, Building owners and other relevant people in the building process such as BIM or information managers and IT related functions.

The skill level of this group is important when designing tools that are usable. This will be explored in the comparison of existing tools in the related work section. An important consideration is that while building professionals are trained to work with complex and advanced software tools they are usually not trained in writing computer programs.

4.1.2. Extensibility & Adaptability

It is important for tools to be extensible when being used in the building process. Building projects vary greatly in scope, character and stakeholders. For example a large residential project requires a very different approach than a stadium construction project, some projects have challenges that resolve mainly about a It is ideal to be able to use the same tools across these projects, so that the building professional can spend the minimal amount of time learning new tools.

Therefore it is important for the tools to be extensible so that they can be adapted to
the project at hand. This extensibility can be achieved by allowing a user to add new functionality to the tool. The skill level of the user is very important in this. For this project we discern four skill levels below, ranging from most to least technically skilled:

1. **Programmer.** A programmer is able to create completely new functionality by creating new functions directly in the code of the tool. To enable this the tool will be released open source, the broader programmer community will then be able to extend it.

2. **Process manager.** An experienced user able to create new functionality based on existing components but is also able to package this functionality so that it can be shared with other users and across different projects.

3. **Building professional.** A regular user is able to create new workflows and connect BIM services using the components provided. By arranging the components in different configurations the tool can be adapted to the project at hand.

4. **Project participant.** A novice user can use an existing configuration on a project that has been provided by a regular user and change input variables of the workflow so that different scenarios can be tested and the tool can be used without in depth knowledge of how the configuration was created.

4.1.3. Open

In order to be extensible and future proof it is important that the tool is open, and works will with open standards. There has been a lot of discussion about interoperability in the building industry as of late, especially regarding recent BIM developments. Using open standards allows companies to efficiently collaborate and communicate, this is key to the goal of this research.

“Interoperability in BIM can contribute to efficiency value levels, through supporting communication and coordination interactions between participants within BIM projects. If higher levels of interactions between participants emerge (e.g., through full 3D BIM cooperation), companies in building projects will likely obtain differentiation value levels, where higher cost benefits and less risk are likely to be the outcome.”

---

The developed tool, the techniques used and related projects that are connected through the tool are using open standards. Many open source solutions exist and increasingly more tools are using open standards such as IFC, mvdXML, Bimsie, BimQL and IfcOpenShell. An overview of the related projects is given in paragraph 7.2.
5. Related work

5.1. Introduction

A wide range of tools have been developed to work with Building Information Models and. In this chapter the tools that are available to work with BIM data and extract data from building models sourced from different vendor packages are categorized. Furthermore existing research on linking BIM models and querying them is evaluated to build a reference framework for evaluating the tools that is introduced in this research.

Point solutions

Often these solutions are closed source and are highly specialized. For example coordination tools like Solibri Model Checker\(^\text{15}\), Navisworks\(^\text{14}\) and BIMsight\(^\text{15}\) have a focus on combining different files, facilitation collaboration in the model, reporting and rule-checking. These tools usually lack in model output possibilities. Queries, rules and query output are saved in a proprietary format, which makes it harder to embed these tools in the building process. Furthermore most of these tools have a fixed UI created for their purposes, they lack in flexibility. Advantage of these tools is that their usability is very good, and therefore they are widely used by building professionals.

Query language approaches

By creating a so called Domain Specific Language (DSL) the complexity of formulating a request for certain building data from a model is reduced. This approach allows great flexibility and composability and is used often in the computer science domain. When applying this approach to the building industry domain typically an existing query language from another domain is used for querying building model data. Examples of


this approach are SQL, SPARQL, BimQL\textsuperscript{16} and Gremlin. Applying these languages can yield good results and powerful tools, with a lot of flexibility, however they often have several usability problems and are therefore not widely used by building professionals. Usability problems in this context mean that using these tools require building professionals to create computer programs, in this case queries. While building professionals are trained to work with complex and advanced software tools they are usually not trained in writing computer programs.

**Toolkits and libraries**

In the third category are toolkits, libraries and applications specifically created for the building industry. There is a lot of variation among these tools, examples are BIMServer\textsuperscript{17}, Jotne EPM\textsuperscript{18}, JSDAI\textsuperscript{19} and IfcOpenShell\textsuperscript{20}. Some of these tools are server based applications, like BIMServer and Jotne EPM. The BIMServer offers limited support for querying, requiring building industry professional to create Java programs. Jotne EPM is a large, complex closed source tool. JSDAI and IfcOpenshell are taking a library approach, making it easier for computer programmers to access building model data by providing a higher level of abstraction when creating programs that work with building models. In general querying for building model data with these tools suffers from either usability problems or a closed source approach.

5.1.4. **BIM Service interface exchange (BIMSie)**

The Bimsie service interface exchange standard is a standard API for BIM web services. It is designed to connect BIM web services in an automated fashion. The goals of the project are to automate interaction between (online) BIM services, giving the industry the possibility to innovate with BIM in the cloud. The standard consists of a set of method calls that can be implemented by BIM web services. The standard is protocol independent, supporting both XML and JSON with the option of adding more. A BIM web service can implement any subset of the 36 Bimsie methods as long as the AuthInterface and ServiceInterface are implemented.


\textsuperscript{18} "EPM Technology - Jotne." 2 Jan. 2015 <http://www.epmtech.jotne.com/>

\textsuperscript{19} "JSDAI." 2003. 2 Jan. 2015 <http://www.jsdai.net/>

\textsuperscript{20} "IfcOpenShell." 2011. 2 Jan. 2015 <http://ifcopenshell.org/>
5.2. Summary of reviewed literature

Papers reviewing the challenges and recent developments in the BIM collaboration area are discussed. The rise of cloud and centralized model server based approaches is a promising direction of research. Using web services or other data sources multi model configurations are now possible, linking data from different domains. The challenge here is to solve interoperability issues and to keep systems usable in projects by emphasizing UI and maintainable by centralizing and facilitating sharing.

First the information delivery manual approach was researched. This is a valuable effort that allows building professionals to create standard processes. The effort has a focus on the collaboration side, specifying which parties should do what. The link to a 3D model is either missing or too implicit and complicated to use in practice, the process of linking to a model through functional parts and MVDs is optimized in the xPPM research project.\(^\text{21}\)

The approach taken in the xPPM research can be expanded in this research to verify if it is possible to model the conversion flows that are implicitly hidden in IDMs, if a project participant has to deliver a model the verification, conversion and storage steps could be explicitly modelled as a dataflow graph on top of an IDM.

Research into the requirements for a BIMserver has revealed that a lot of problems interoperability problems still exists.\(^\text{22}\) Converting to and from different vendor formats, or from and to into IFC. A wide variety of tools is used, not only for modelling but also analysis and querying. The number of conversions required is large and many manual steps are required. Also merging IFC files often involves manual steps, for example positioning models, filtering duplicate data and scaling. Also discussed is a focus on UI and visualization, a dataflow approach can clearly visualize the required steps and intermediate representations.

Finally there is a lack of transparency and configuration when merging models, leading


to data loss, a dataflow with clear and non-destructive steps helps in this case. Also discussed is the ability to link different models together, called federation, this linking step can also be modelled as a dataflow.

Linking models, and data in general, is an interesting topic in its own right. From the cloud and interoperability perspective there are relations to be found with this research project. The advancement of cloud technology and web services alone is not enough to solve the interoperability problems that exist, additional tools are required to link data together. One way is to lift the data into a common format like RDF.

After lifting the data custom applications have to be developed on top of the graph for it to be usable, which is time consuming, also often requirements change between projects, leading to throwaway work. The RDF approach is very powerful though, and a lot of efforts have been made to lift building data to this format. It is therefore interesting to see if the more generic lifted graphs can be used in the dataflow model. The dataflow can then act as a glue between the different graphs and make the time consuming step of developing custom applications easier.

The method of using a dataflow to describe a workflow and transformation process is described by Vasenev, Hartmann, et. al. Contrary to earlier approaches a more ad-hoc way is chosen. The dataflow is used as a framework upon which applications are built. However the dataflow itself is merely used as a model, not as the implementation itself. This suggests that a closer coupling between the dataflow and the implementation is a direction to explore. Generating the working program directly from the dataflow. The case study in this article is very detailed and therefore a good candidate to apply to the approach I will take.

An important part of processing model data is selecting parts of a model by using queries. Approaches for querying are discussed in the BimQL research. The article highlights that different stakeholders with varying skillsets are involved in the building design process. A differentiation between the different skill levels is necessary to make an approach accessible for all groups.

The fact that the article acknowledges the need for more ad hoc per project querying is also important. Composing queries should be quick and easy to set up. The per project ad hoc needs of the building industry are large, projects are different, sites are different and usually even the primary stakeholders vary between projects. It is important to provide tools that can help with the ad hoc needs from designers and engineers during the building design and construction process. Finally querying across different repositories is cited a future work, this is a challenges that this research can potentially offer a solution for, by providing a dataflow graph that uses BimQL queries as components.

Performance is cited as an important issue in the BimQL article. The dataflow programming could help with this. Advances in dataflow programming languages\textsuperscript{26} discussed the origins of dataflow programming. The initial goal of it was to address performance issues with more traditional sequential computing. It is an interesting topic to see what performance gain can be realized by dataflow graphs in the AEC domain.

Granularity of the dataflow elements became an issue with optimizing the performance and together with multithreaded programming approaches this led to a variant of dataflow programming called Flow Based Programming\textsuperscript{27}. It simplifies dataflow programming even further by allowing users to compose low level sequential computing blocks into higher level structures. The article cites empirical evidence of the productivity gains when using this approach.

Also supporting the approach of applying dataflow to the AEC domain is the wide variety of commercially successful dataflow environments in other domains. Such as LabView, Prograph and many ETL solutions.


6. Dataflow programming

6.1. Introduction

Dataflow programming is a programming paradigm that models a computer program as a directed graph. In this graph the data flows over the edges from one node to the next node. Where each node can perform operations to the data. Typically a node does not have any so called “side-effects” where it performs an operation that has no effect on the data that it outputs but on other data that is not represented in the graph. This has some similarities to the functional programming paradigm.

The original motivation for research into dataflow programming was the exploitation of massive parallelism in computer programs using computer processors. Originally it was a criticism against traditional Von Neumann processor architecture, where a central bus is the bottleneck of a CPU. New and more modern architectures have superseded this model and therefore most of the architectures pursued today are a hybrid between traditional dataflow programs and Von Neumann processing\(^\text{28}\).

In a pure dataflow model edges represent data dependencies between nodes, and nodes are the primary instructions. Data flows along edges which behave like unbounded FIFO queues. When the program begins data is put on special activation nodes, triggering the rest of the program. When all inputs of a node have data on them a node becomes fireable and will process the input and put data on it’s output edges. The key advantage of this is that more instructions can be executed at once. If a node has two edges leaving from the same output port then data is copied, different types of special gate nodes can be used to combine data again, for example switch or merge gates that merge data based on certain conditions.

---

Dataflow programs have two approaches to control execution. The data driven approach where execution is dependent on the availability of data, a node is inactive while data is arriving and an overall management device notifies and fires nodes when their data has arrived. The second approach is demand driven, where a node is activated when it receives a request for data from its output arcs. The demand driven approach can lead to optimizations because nodes don’t have to execute if they are not used.

Core features of dataflow languages are:

1. Freedom from side effects
2. Locality of effect
3. Data dependencies equivalent to scheduling
4. Single assignment of variables
5. An unusual notation for iterations because of 1 and 4
6. Lack of history sensitivity in procedures

There was a lot of growth in DFVPLs (dataflow visual programming languages) in the 90s. With many commercial applications such as Labview and Prograph. DFVPLs are no longer primarily about parallelism but more software engineering in general. DFVPLs are currently the main development in dataflow programming languages. It is clear that the graphs have value of themselves for the programmer. Graphs allow easy communication, there are significant advantages in visual syntax and the fact that
several commercial efforts have been widely successful add weight to this case.

Recent DFVPLs often came from industry needs instead of research, they are most successful in narrow application domains where data manipulation is the foremost task. LabView being one of them. A case is discussed where a company using LabView report significantly faster development time and improved communicated compared to program development in C. The case is also made for dataflow programming as a coordination language, where the nodes are the computations and the graph specifies the order and relations between them. Dataflow is a natural coordination language. Morrison describes a concept where nodes are built in arbitrary programming languages and the programmer arranges them in a dataflow.

Dataflow programming environments are widely used across a varying number of domains. Some examples are listed on the following page.

- **Labview**[^29]: Measurement and control system for engineers, nodes can represent sensors or transformations on sensor data
- **SQL Server**[^30]: Nodes can represent transformation on data that is extracted from a database, can also be used to visually create complex SQL queries
- **Unreal Kismet**[^31]: A system to program the Unreal 3d game engine, nodes can represent entities in the game world and interactions between them
- **Noflo**[^32]: A more general web based dataflow environment where nodes can represent elements on a web page or user interactions with them
- **Blender**[^33]: As part of the 3d modelling software a dataflow environment is used to generate and manipulate geometry, textures and animations.

In all of these solutions the dataflow programming environment is being used to decompose complex operations into a visual representation of many smaller simple operations. This is also done in the AEC domain, which will be discussed in chapter 6.3.

6.2. Examples of commercial dataflow environments

Labview

SQL Server

Unreal Kismet

Noflo

Blender
6.3. Dataflow programming in the AEC domain

Dataflow programming can be applied to a wide variety of domains, therefore it is interesting to explore existing solutions in the AEC domain as well. Two main approaches were found where dataflow programming is used in the AEC domain with several plugins and extensions available for either approach.

6.3.1. Grasshopper

Grasshopper\(^{34}\) is a visual dataflow programming environment tightly integrated with Rhino’s 3-D modeling tools. Unlike the scripting languages integrated into Rhino, Grasshopper requires no knowledge of programming or scripting, but still allows designers to build form generators.

![Fig. Grasshopper screenshot\(^{35}\)](http://www.grasshopper3d.com/)


Grasshopper is a graphical algorithm editor that is integrated with Rhino3D’s modeling tools. You use Grasshopper to design algorithms that then automate tasks in Rhino3D.

Grasshopper consists of two primary types of user objects: parameters and components. Parameters store data, whereas components perform actions that result in data. The most basic way to understand Grasshopper is to remember that we will use data to define the inputs of actions (which will result in new data that we can continue to use).

Parameters store the data - numbers, colors, geometry, and more - that we send through the graph in our definition. Parameters are container objects which are usually shown as small rectangular boxes with a single input and single output. We also know that these are parameters because of the shape of their icon. All parameter objects have a hexagonal border around their icon. Components perform actions based on the inputs they receive. There are many types of components for different tasks.\(^{36}\)

![Image of Grasshopper subtract node]

Archsim\(^ {37}\)

Archsim Energy Modeling is a plugin that brings EnergyPlus simulations to Rhino/Grasshopper and thus links the EnergyPlus simulation engine with a parametric design and CAD modeling environment. It is an example of one of the ways to extend

\(^{36}\) "Mode Lab | Grasshopper Primer." 2014. 31 Jan. 2015 <http://modelab.is/grasshopper-primer/>  
Grasshopper through programming new components that can be used in a graph configuration.

Geometry Gym

"Geometry Gym provides OpenBIM software tools and support for Architects and Engineers and those in the Construction Industry. Primarily these tools target streamlined workflow processes for improved productivity and designs. Powerful Parametric and Generative modelling of projects is enabled by utilizing plug-ins for Grasshopper3d and Rhino3d. Generative BIM model exchange is provided using a combination of OpenBIM formats (primarily IFC) and direct API interaction to popular software including Revit, Archicad, Digital Project and Tekla. Structural Analysis models can be exchanged with many popular analysis software."

Geometry Gym is another plug-in for Grasshopper that extends it with new components that allow for new functionality. Geometry Gym is similar in goals to this research paper and has created functionality to combine different vendor tools to create

streamlined workflows.

6.3.2. Dynamo

“Dynamo is a visual programming tool that aims to be accessible to both non-programmers and programmers alike. It gives users the ability to visually script behavior, define custom pieces of logic, and script using various textual programming languages.”

Dynamo is a tool built on Autodesk Revit, a CAD package for creating building information models. It is similar to Grasshopper in that it is a visual dataflow programming environment on top of an existing CAD package.

The structure of components and connectors is similar to Grasshopper and other dataflow environments. Dynamo is primarily used for geometry generation and has integrations with several other tools such as excel and other Autodesk products.

6.3.3. Conclusion

Dataflow programming is a concept that is already being used in the AEC sector. It is implemented on top of two existing CAD packages, Grasshopper and Revit, by two different vendors. Both projects are actively developed and can be extended by plug-ins to provide for new functionality. Dynamo is distributed as an open source project and therefore allows for further extensibility and adaptability.

There are however a few limitations to both tools, as described below:

**Tied to a specific CAD package**
Both tools are tied to a specific CAD package and require it to function (Grasshopper or Revit). This makes for more difficult sharing of workflows, as all stakeholders require experience with the specific tool to use the workflows. There are also limitations in file formats, even though both tools allow for importing and exporting to different formats, including IFC, there is often data loss as internally the CAD package’s format is used to do the conversion.

**Difficult to collaborate on workflows**
Both tools are offline, client based tools, this means that it is run on a workstation by a single user, is not accessible over the web, and cannot be exposed as a web service without significant efforts. This makes it less efficient to collaborate, as different stakeholders will need to install the underlying CAD package and the dataflow environment on their machine. Exchanging workflows is currently file based only for both tools, instead of leveraging a server based approach where multiple stakeholders can collaborate through the web.

**Primarily created for generating geometry**
Even though extensions have been developed both Grasshopper and Dynamo are currently primarily used for generating geometry. Many nodes are available for generating geometry and manipulating it. When looking at connecting BIM web services less effort has been made. This extra functionality could be developed in the form of plug-ins with some effort.
7. Framework overview

7.1. Overview

As a solution to connecting a wide array of BIM services currently used we are introducing the IFC Dataflow Framework. This is a framework that allows building professionals to connect different BIM services and provides for a wrapper toolkit to convert existing tools that are not exposed as a web service. The framework allows a building professional to log in and create dataflow graphs that connect different services.

A graph can be of two types, main or sub-graph. A main graph is a program that can be executed by the user in itself and that describes the workflow in which services are connected. Subgraphs also describe this workflow but can only be used in main graphs. They can be used to abstract certain parts of the workflow and they can be re-used across other graphs.

7.1.1. Graph architecture

The dataflow graph that describes the workflow is defined as a bipartite labeled...

---

**Fig. sub graph composition**
graph where the two types of nodes are called actors and links.

A set of nodes connected by edges where the edges have a direction associated with them. A dataflow graph $G$ can be defined as:

$$G = (A \cup L, E)$$

where:

$A = \{a_1, a_2, \ldots, a_n\}$ is the set of actors

$L = \{l_1, l_2, \ldots, l_3\}$ is the set of links

$E \subseteq (A \times L) \cup (L \times A)$ is the set of edges

Actors are commonly referred to as nodes, and represent the operations that are executed while running the graph. The links are placeholders of data values as they are transferred between actors. Edges are the method of communication between nodes, as is common when defining graphs.\(^\text{40}\)

Once an operation has completed it will put its result on the input of all connected output nodes. If all their inputs are filled then they will start to execute, this repeats until the last node is reached. Nodes can have several extra properties, beyond its input and output ports.

An edge can also be connected from the output of a single port to multiple ports on different nodes, the same output data will be sent to all the connected nodes. A port can also be given a specific type, ports can only be connected if they have the same type or if the types are related through type polymorphism.

7.1.2. Node specification

A node can have a separate user interface that allows a user to influence the operation of the node while the graph is running. Once the input ports are filled the user interface component will update itself. Once the user manipulates the UI it can re-trigger the operation, or perform additional operations based on the UI, changing the output. The UI should not block the operation of the graph, therefore there should always be a default option available.

An example of this is the GetProject node (see section 8.3.2), where a Bimsie session is used as input, once the input is filled the relevant Bimsie API will be called to fill a dropdown UI component with the projects from the model server at the given Bimsie url. A user can then select a project and an additional API call is done to fetch the relevant project data based on the selected project. This data is then put on the output port.

The framework also provides a 3d visualization canvas that can display results of operations immediately. The 3d visualization automatically updates as new results come in, providing an interactive workflow that is crucial for building professionals. There are two conditions a node to add its results to the 3d canvas, first it needs to have a model type as output and second it needs to be specified that the hasModel property refers to a Model View.
A eval function can also make calls to external services. This is done through doing an HTTP call from the client, the preferred exchange format is JSON, more on the data exchange with external services is covered later in this chapter.

<table>
<thead>
<tr>
<th>Type</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any</td>
<td>Superclass of all types, can also be used in ports if any type is allowed</td>
</tr>
<tr>
<td>String</td>
<td>UTF-8 string</td>
</tr>
<tr>
<td>Number</td>
<td>A Javascript Number type</td>
</tr>
<tr>
<td>Bimsie Session</td>
<td>Reference to a Bimsie compliant model server and a valid session token, can be used for other Bimsie operations</td>
</tr>
<tr>
<td>Bimsie Project</td>
<td>Reference to a Bimsie compliant project</td>
</tr>
<tr>
<td>Model</td>
<td>A representation of a building model, contains a unique identifier that can be used to fetch the model from a Bimsie compliant model server. A model can originate from an existing revision on the model server or from a temporary revision that is the result of another operation.</td>
</tr>
</tbody>
</table>
7.1.3. Bimsie types

The Bimsie types are introduced to simplify working with Bimsie compliant model servers. The types Session and Project correspond to the concepts of Authentication and Project as they are defined in the Bimsie standard. The session is used to store the url to the Bimsie model server and a valid session token. The Project holds a reference to a Bimsie Project type.

Some operations need to store temporary model data on the model server. There is no concept for this in the Bimsie standard, so the concept of a working project is introduced. This project is used to store temporary model data on the model server in the form of a Revision in this Project.

The concepts of temporary model data and a Bimsie Revision have been merged to further simplify the dataflow interface in the form of a Model type. The Model type can refer to any Bimsie Revision both in the working project and in any other Bimsie Project. Because building models can become very large in size it is not recommended to transfer the models to client often. Therefore a Model type only contains a reference to a Revision, and not the actual model data. The model data can be downloaded using the Bimsie Session when needed by the dataflow environment.

To significantly decrease the amount of dataflow nodes required to model a workflow the Bimsie datatypes form an inheritance structure. This requires a Model to also include the data from the Project and Session, so that any operation that uses this Model as input data also has access to the Bimsie session. Otherwise the user has to manually connect these data types to all the operations that require the data.

An example of how this is implemented on the Open Source BIMserver is presented in chapter 7.2.2. The figure below illustrates how the datatypes flow through the different nodes in this use case. Initially a Bimsie Session is retrieved in the Get Bimserver node, this data is sent to the Get Project node, which retrieves the project list and allows the user to select a project from the model server, it sends this project data to the connected Get Revision node, the Bimsie Session is included in this data. The Get Revision node adds the revision to this information in the same manner. Then the Query node generates a new model and sends it to the Merge node. This process is executed in parallel on another Bimsie model server. The results of both sequences are sent to the Merge node. This node uses the first model server to do its operations on
and thus sends a third model, containing the model data from both Bimsie model servers to the final Query node.

Fig. Example merge dataflow with Bimsie datatypes

7.1.4. Data transfer

Data is transferred when the output data of a node needs to serve as input for the next node. All the data is kept on the client that is running the dataflow graph. Data is specified according to the types and is transferred in a JSON data structure\textsuperscript{41}. JSON is a free form data structure that is used by many web services. It is a nested set of key-value pairs with several basic datatypes like numbers, strings and arrays provided. JSON was chosen because it is very flexible, and can therefore be easily implemented or retrofitted on an existing service using a wrapper. In this framework the client runs in a web browser, where JSON can be efficiently parsed. The Bimsie API also provides JSON as a transfer format, which allows for efficient integration.

Some operations can be done on the client, using the data that is served by the client on the input ports, in this case no additional data transfer is done. Most operations however interface with a web service. Interfacing with web services is done over the HTTP protocol through a REST based API. The HTTP protocol requires the web services to expose their endpoints over HTTP URIs (Uniform Resource Identifier). The services implemented in this framework all adhere to principles of RESTful web services\textsuperscript{42}. This is a loosely defined set of guidelines according to which APIs can be designed. The data is also transferred in the JSON data format between external services and the client.

7.1.5. Servers & distribution of workload

The workload can be divided between different servers to allow for efficient computation, even with larger models or many operations. Components can communicate with an external Bimsie compliant model server or other services, so while the data is transferred through the client application the operations can be performed on other servers “in the cloud”. The environment itself is also powered by an internal BIMserver that does not block the main operations that run in the client.

7.1.6. Existing services / wrapper

Existing services might not be accessible in the way that the dataflow environment expects. In this case a wrapper needs to be created to convert the existing service into a

---

web service. A method for doing this is proposed here. It is not feasible to create wrappers in an automated way because different services and tools need to be called using different methods. For example a tool might be an executable, shell script, Java program or SOAP service.

For this reason it is required to program the wrapper so that it connects to the existing services, after the wrapper is created no programmer is required to call the service, it can be used by adding a component in the dataflow environment.

There 3 parts to a wrapper service, wrapper program, API, dataflow node.

### 7.1.7. BIM Service API

To facilitate the creation of new service wrappers a lightweight variant of the Bimsie API is introduced. When using the Bimsie API services are implemented on top of a Bimserver. The following calls should be implemented by a wrapper service.

<table>
<thead>
<tr>
<th>Method</th>
<th>Parameters</th>
<th>HTTP Responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>AuthInterface loginUserToken</td>
<td>&lt;token&gt;</td>
<td>200 &lt;token&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>400 Invalid token</td>
</tr>
<tr>
<td>AuthInterface getAccessMethod</td>
<td>-</td>
<td>200 [&lt;access_method&gt;]</td>
</tr>
<tr>
<td>Bimsie1ServiceInterface.checkin</td>
<td>&lt;model&gt; &lt;additional_data&gt;</td>
<td>200 &lt;checkin_uuid&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>400 Invalid data</td>
</tr>
<tr>
<td>Bimsie1NotificationRegistryInterface.getProgress</td>
<td>&lt;checkin_uuid&gt;</td>
<td>200 &lt;progress&gt;</td>
</tr>
<tr>
<td>Bimsie1ServiceInterface.getDownloadData</td>
<td>&lt;checkin_uuid&gt;</td>
<td>200 &lt;model&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>400 Invalid model</td>
</tr>
<tr>
<td></td>
<td></td>
<td>500 Processing error</td>
</tr>
</tbody>
</table>

The listed types should be implemented as follows

<table>
<thead>
<tr>
<th>Type</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;token&gt;</td>
<td>String with the token for this session</td>
</tr>
<tr>
<td>&lt;string&gt;</td>
<td>UTF-8 string</td>
</tr>
<tr>
<td>&lt;access_method&gt;</td>
<td>Access method that can be used to interact with the service, currently either “JSON” or “XML”</td>
</tr>
<tr>
<td>Tag</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>&lt;uuid&gt;</td>
<td>Unique identifier for the checkin</td>
</tr>
<tr>
<td>&lt;model&gt;</td>
<td>Building model serialized as Ifc and base64 encoded</td>
</tr>
<tr>
<td>&lt;checkin_uuid&gt;</td>
<td>Unique identifier for the current checkin</td>
</tr>
<tr>
<td>&lt;additional_data&gt;</td>
<td>JSON or XML data containing additional data that the wrapper script might need</td>
</tr>
<tr>
<td>&lt;progress&gt;</td>
<td>Enum that indicates the progress, possible values:</td>
</tr>
<tr>
<td></td>
<td>- UNKNOWN</td>
</tr>
<tr>
<td></td>
<td>- STARTED</td>
</tr>
<tr>
<td></td>
<td>- AS_ERROR</td>
</tr>
<tr>
<td></td>
<td>- FINISHED</td>
</tr>
</tbody>
</table>

A standard exchange where the dataflow environment connects to the service, performs a checkin and then processes the result is displayed below:

![Diagram of dataflow process](image)

The wrapper program has to call the existing service and process the output into a standard format. Usually there is also some preliminary work that is needed, for example converting a model and placing it into the local repository so that the service can use this information.
7.2. Open source projects

7.2.1. Flood

“Flood is a dataflow-style visual programming environment based on Scheme written in JavaScript. flood runs in a browser and as a standalone application on all platforms via node-webkit.”

The project is an open source visual programming environment which can be freely used. This project was adapted and expanded to become the IFC dataflow client. It is written in node.js and html/javascript and consists mostly of the browser client and a small server side application to persist the graph state.

The project was altered to provide for more nodes and to change the semantics of several nodes to accommodate for rendering models into the 3d canvas. Flood contains basic types for the representation of nodes, edges, etc. For IFC dataflow a model subtype was created with the standard Flood Node class as its parent.

7.2.2. BIMserver

The Open Source BIMserver is used as an internal backend service for doing most of the

---

operations on building models. It is also used as an external service that can be queries and interacted with by the user through nodes that correspond roughly to the Bimsie API\textsuperscript{44}. The Bimsie API is a standard application interface for working with model servers. The concept of connecting different BIM services through this API has been introduced as Federated BIM by Leon van Berlo.\textsuperscript{45}

The IFC dataflow project can be used to connect existing model server using the Bimsie API so that they can share data, provide input for other operations, or to store the results of operations. Examples of operations are download and uploading models to the server, querying them for information or combining several models by merging them.

The BIMserver itself has not been modified but the semantics of the API have been changed and simplified to improve the usability of the dataflow interface. The first concept that has been simplified is the use of serializers. When the Bimserver works with model data it specifies how to convert the information from and to its internal format by using serializers. Examples are the IfcSerializer and the ColladaSerializer. When using the dataflow approach the output of an operation can flow directly into the input of another operation, for example a query and a merge operations can be chained directly. Only at the end it is relevant in which format the data is downloaded. Therefore the concept of serializers can be abstracted from the API when using dataflow concepts. The IFC dataflow graph will internally add the required serializers when using the Bimsie API.

Another change in semantics is that the result of various BIMserver operations is often the serialized data. This means that for example after doing a query the result will be the raw IFC data, or any other serialized format of the result. Subsequent operations such as merge, or doing another query all have revisions as input, so in this case the model needs to be checked in again before it can be re-used. This concept has been simplified by creating a concept called ‘model’ which is a revision in any form on any BIMserver. The model is used as input and output of all operations, it contains information such as the BIMserver, project and revision information, see BIMserver types.

\textsuperscript{44} "BIMSie - National Institute of Building Sciences.” 2013. 12 Jan. 2015 <http://www.nibs.org/?page=bsa_bimsie>

\textsuperscript{45} "Federated BIM and BIMSie Léon Berlo 602 views - Slideshare." 2014. 21 Jan. 2015 <http://www.slideshare.net/berlotti/federated-bim-and-bimsie>
Among other simplifications this leads to a decrease of the number of visible nodes to arrange by the user. For example in the case of a query being done over a merge model that originates from two different model servers. In this case the simplified version of the graph has 8 nodes, which results in 26 Bimsie API calls.

![Simplified dataflow graph for merge and query example](image)

**Fig. Simplified dataflow graph for merge and query example**

![Underlying graph of Bimsie API calls](image)

**Fig. Underlying graph of Bimsie API calls**

### 7.2.3. BimQL

“BimQL (Building Information Model Query Language) is an open, domain specific query language for Building Information Models. The query language is intended for
selecting and updating data stored in IFC models and it is currently implemented on top of the BimServer.org platform...”

BimQL has been integrated into the project as a Query node, which takes a string with the BimQL query as input. BimQL is currently being exposed through the Open Source BIMserver. It is integrated as a plug-in into the BIMserver, which means it is also exposed through the Bimsie API. The BIMserver that is used for doing the query is always the source BIMserver that contains the model that needs to be queried.

7.2.4. mvdXMLChecker

MvdXML is a “Specification of a standardized format to define and exchange Model View Definitions with Exchange Requirements and Validation Rules”

The mvdXML standards allow a user to describe validations in the form of a XML file. These validations can then be tested against a building model. Objects of the building model that do not pass the validations can be raised as an issue. An implementation of the mvdXML standard is done by the mvdXMLChecker project. The project comes in the form of an executable jar file that takes the IFC schema, a mvdXML file and a target building model in the IFC format as input. The output is a list of IFC guids that do not conform to the given validations.

The mvdXMLChecker project has been used as-is, however a wrapper server has been created so that the executable can be called as a web service over HTTP.

---

The proposed framework utilizes a multi-tier client-server architecture, built around the principles of Service Oriented Architecture (SOA). SOA is defined by the World Wide Web Consortium (W3C) as "a set of components which can be invoked, and whose interface descriptions can be published and discovered"\(^{49}\). The purpose is to define services that can easily interact with each other, and therefore to make it possible to have services be the fundamental building block by which applications are created. Interoperability is emphasized with SOA, and it combines the capacity to invoke remote objects and functions with standardized mechanisms for execution.

The method by which services can be composed has been described earlier, the framework architecture consists of the following main parts:

<http://www.w3.org/TR/ws-gloss/>
The first layer of the framework is the Dataflow client. This is what the user will interact with when working with the tool. It is an HTML5 application that runs in the user’s web browser using Javascript. The main function of the client is to provide an intuitive user interface that allows the user to create dataflow graphs. Earlier this has been described as the dataflow visual programming environment. The dataflow client also needs to keep state, such as the positions and initial values of components, and the graph configurations that a user has created earlier. For this reason there is also a dataflow server application that has a database (MongoDB) to keep this state. This allows for persistent user data and sharing graphs between different connected clients.

Part of this is back-end is also a ‘native’ BIMserver that is required for the application to function, this BIMserver handles built-in components such as BimQL that depend on a BIMserver to function, as well as handling the conversion of file formats for the 3d visualization. This 3d visualization is the final important component of the dataflow client, it allows for an interactive workflow and viewing of intermediate components.

7.3.1. Implementation notes

The client uses several HTML5 web workers to control each graph. Web workers define an API that allows Web application authors to spawn background workers running scripts in parallel to their main page. This allows for thread-like operation with message-passing as the coordination mechanism\(^{50}\).

The web workers coordinate the execution of the dataflow graph. Determining in which order components need to be executed. This is done by traversing the connections between components until those components with no dependencies are found, these can be executed, the data is transferred to any connected components and then the process repeats until every node that is able to execute is executed. On any data change or addition this process is repeated so that new inputs are immediately processed. Only the components that are affected by the changed data will re-execute.

Several components depend on external services, these services can be existing HTTP services such as a BIMserver, or existing tools that are wrapped by a service wrapper so that they are exposed over HTTP. The web workers can query these services and await results, these results can then be processed by the components like described above.

---

8. Dataflow nodes

8.1. Introduction

The possibilities of the dataflow environment are primarily defined by the components that are available. For the prototype a set of basic components was developed that demonstrates how common use cases can be implemented using dataflow programming. The set consists of 7 generic nodes that can be used to perform basic computations and provide input to other nodes, 5 Bimsie nodes that provide features that are exposed by using various Bimsie APIs and 2 Extension nodes that rely on other services such as the mvdXMLChecker or BIMserver plugins such as BimQL.

Nodes are described as HTML/Javascript components, where the UI of the node is represented as HTML and all logic is described in Javascript. Creating a new node is straightforward and requires implementing a standard API that is described in the previous chapter.

At least a name, inputs, outputs and an eval function needs to be defined for the node to exist. The eval function is what gets executed when all the input ports have data. Extra methods and properties can be defined on a component if more interaction with the user is required, for example filling a UI dropdown with values whenever the input ports are satisfied and then putting the result of the the selection from the UI on the output port.

Every component has input and output ports. In addition to the names of the ports they also can hold a specific type of data. This makes it easier to work with the tool because it is harder to create incompatible connections. The user interface will block the user from creating a connection if the types do not match.
## 8.2. Generic nodes

### 8.2.1. Input string

<table>
<thead>
<tr>
<th>User interface</th>
<th><img src="image1.png" alt="Image" /></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>This node can be used to input a string into the graph from the UI. The node provides for an input text field, the data entered can be passed on to other nodes.</td>
</tr>
<tr>
<td><strong>Input type</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Output type</strong></td>
<td>String</td>
</tr>
<tr>
<td><strong>Settings</strong></td>
<td>None</td>
</tr>
</tbody>
</table>

### 8.2.2. Input number

<table>
<thead>
<tr>
<th>User interface</th>
<th><img src="image2.png" alt="Image" /></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>This node can be used to input a number into the graph from the UI. The node provides for a slider and a text field, the data entered can be passed on to other nodes.</td>
</tr>
<tr>
<td><strong>Input type</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Output type</strong></td>
<td>Number</td>
</tr>
<tr>
<td><strong>Settings</strong></td>
<td>Step, Minimum value, Maximum value</td>
</tr>
</tbody>
</table>
8.2.3. Show output

<table>
<thead>
<tr>
<th>User Interface</th>
<th>![Image](182x562 to 334x670)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>This node can be used to show the source output of any other node. In case of an object the object will be printed out as JSON.</td>
</tr>
<tr>
<td><strong>Input type</strong></td>
<td>Anything</td>
</tr>
<tr>
<td><strong>Output type</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Settings</strong></td>
<td>None</td>
</tr>
</tbody>
</table>

8.2.4. Eval

<table>
<thead>
<tr>
<th>User Interface</th>
<th>![Image](182x303 to 326x374)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>This node takes source input and applies the function described in this input as text by the user and sends the result of the function to the output nodes.</td>
</tr>
<tr>
<td><strong>Input type</strong></td>
<td>String</td>
</tr>
<tr>
<td><strong>Output type</strong></td>
<td>Anything</td>
</tr>
<tr>
<td><strong>Settings</strong></td>
<td>None</td>
</tr>
</tbody>
</table>
8.2.5. Add

**User Interface**

![Add Node Image]

**Description**
This node takes two input numbers, adds them together and sends the resulting number on the output nodes.

**Input type**
A : Number, B : Number

**Output type**
Number

**Settings**
None

8.2.6. Multiply

**User Interface**

![Multiply Node Image]

**Description**
This node takes two input numbers, multiples them and sends the resulting number on the output nodes.

**Input type**
A : Number, B : Number

**Output type**
Number

**Settings**
None

8.2.7. Subtract
<table>
<thead>
<tr>
<th><strong>User Interface</strong></th>
<th><img src="image.png" alt="Subtract node diagram" /></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>This node takes two input numbers, subtracts B from A and sends the resulting number on the output nodes.</td>
</tr>
<tr>
<td><strong>Input type</strong></td>
<td>A : Number, B : Number</td>
</tr>
<tr>
<td><strong>Output type</strong></td>
<td>Number</td>
</tr>
<tr>
<td><strong>Settings</strong></td>
<td>None</td>
</tr>
</tbody>
</table>
8.3. Bimsie nodes

8.3.1. Get bimserver

User Interface

![User Interface Diagram]

Description
This node takes three input strings as input, email, password and url. The url should be the Bimsie service’s main url. This is used to perform the following Bimsie operations:

1. AuthInterface.Login
2. Bimsie1ServiceInterface.getSerializerByName (Ifc)
3. Bimsie1ServiceInterface.getDeserializerByName (Ifc)
4. Bimsie1ServiceInterface.getQueryEngineByName (BimQL)
5. Bimsie1ServiceInterface.getSerializerByName (Collada)
6. Create a bimflow working project or retrieve an existing working project reference

The resulting data is wrapped into Bimsie Session object and passed on to the output nodes.

Input type
Email : String, Password : String, URL : String

Output type
Bimsie Session

Settings
None

8.3.2. Get project

User Interface

![User Interface Diagram]

Description
This node takes a bimserver as input node and then performs a get project operation on the bimserver. The node then renders a dropdown with the available projects from the given bimserver.
The user can select a project from the list and this project is passed on to the output nodes as a Project datatype.

<table>
<thead>
<tr>
<th>Input type</th>
<th>bimserver : Bimsie Session</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output type</td>
<td>Bimsie Project</td>
</tr>
<tr>
<td>Settings</td>
<td>None</td>
</tr>
</tbody>
</table>

### 8.3.3. Get revision

**User Interface**

```
project  asd  model
```

**Description**

This node retrieves a specific revision from the given project. The revisions are retrieved using the project and bimserver references given.

The list of all revisions of the project (and all its subprojects) is rendered into the UI. After the user selects a revision from the list the information is wrapped in a Model object and passed on to the output nodes.

<table>
<thead>
<tr>
<th>Input type</th>
<th>project : Bimsie Project</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output type</td>
<td>model : Model</td>
</tr>
<tr>
<td>Settings</td>
<td>X Offset, Y Offset, Z Offset, Visibility</td>
</tr>
</tbody>
</table>

### 8.3.4. Merge models

**User Interface**

```
model1 MergeModels model
model2 ...
```

**Description**

This node merges two given models into a single model containing all the geometry and other information from both the models.

The merge is done on the model server of the first model given, and this
bimserver is passed on with the model to the output nodes.

<table>
<thead>
<tr>
<th><strong>Input type</strong></th>
<th>project : Bimsie Project</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Output type</strong></td>
<td>model : Model</td>
</tr>
<tr>
<td><strong>Settings</strong></td>
<td>X Offset, Y Offset, Z Offset, Visibility</td>
</tr>
</tbody>
</table>

N.B.: when merging more than 1 model this can be done by chaining merges into a stepwise fashion, where the output of the first merge node is the first input of the second node, etc.

8.3.5. Get object info

<table>
<thead>
<tr>
<th><strong>User Interface</strong></th>
<th><img src="image" alt="Diagram" /></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>This node retrieves the object information from a given model. The object information is represented as a json dictionary and contains a basic quantity report that has quantities based on the IFC element types such as IfcWallStandardCase, IfcWindow, etc.</td>
</tr>
<tr>
<td><strong>Input type</strong></td>
<td>model : Model</td>
</tr>
<tr>
<td><strong>Output type</strong></td>
<td>report : Javascript Object</td>
</tr>
<tr>
<td><strong>Settings</strong></td>
<td>None</td>
</tr>
</tbody>
</table>
8.4. Extension nodes

8.4.1. BimQL query

<table>
<thead>
<tr>
<th>User Interface</th>
<th>![Diagram of BimQL query node]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>This node executes a BimQL query on the given model. The IFC data in the model that matches the given query will be sent to the output nodes as a new model. The BimQL query is given as a string. BimQL is an open source query language that makes it easier to query IFC models, it is embedded into the BIMserver project.</td>
</tr>
<tr>
<td>Input type</td>
<td>model: Model, query: String</td>
</tr>
<tr>
<td>Output type</td>
<td>model: Model</td>
</tr>
<tr>
<td>Settings</td>
<td>X Offset, Y Offset, Z Offset, Visibility</td>
</tr>
</tbody>
</table>

8.4.2. mvdXML

<table>
<thead>
<tr>
<th>User Interface</th>
<th>![Diagram of mvdXML validation node]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>This node applies an mvdXML validation to the given model. Objects that have an issue according to the given mvdXML file will be highlighted in red in the 3d visualization. The resulting model will be sent to the output nodes. The node takes a model and</td>
</tr>
</tbody>
</table>

---

an mvdXML validation file as input, the mvdxm input is a url where the full xml file can be downloaded.

It uses the open source mvdXMLChecker52 with a custom mvdXMLServer that wraps it into a server based component.

<table>
<thead>
<tr>
<th><strong>Input type</strong></th>
<th>model : Model, mvdxm : String (url)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Output type</strong></td>
<td>model : Model</td>
</tr>
<tr>
<td><strong>Settings</strong></td>
<td>X Offset, Y Offset, Z Offset, Visibility</td>
</tr>
</tbody>
</table>

9. Example scenario

9.1. Introduction

In order to validate the approach of dataflow programming to connect BIM services, an example scenario was designed that combines different open source BIM services to create a workflow that could occur in the daily practice of a building professional. Using this example we demonstrate how the framework works in practice and we can validate if the problems stated are resolved to a reasonable degree. We can also conduct performance tests of the scenario and explore opportunities for improvement.

In this example scenario we have an Architectural practice, a MEP design practice, and the building owner. All working on the design of a small duplex house. The model used is the “Duplex Apartment” model provided by the National Institute of Building Sciences as part of the Common Building Information Model Files and Tools\(^53\). The HVAC and Architectural sub models are used.

The architect, MEP designer and building owner’s office have their own model servers running, which they use for both internal and external collaboration. It was decided during the project planning phase that all final BIM data would be made available to the building owner’s model server.

In this scenario the building owner wants to perform a validation on the latest building model to make sure that it conforms to certain parts of the current building codes and regulations. For example checking if certain doors are made of fire proof material. For this purpose the building owner wants to pull the latest model data from the model servers of the MEP designer and architect. Not all the data that is generated is needed for doing the building code validation, so that building owner wants only to pull a subset of the data, this is done using the BimQL language. After pulling the relevant data both sub models need to be merged together to do a validation that requires both the input from the MEP designer and the architect, this is done using the building owner’s model server, a tool they are already familiar with. Finally the validation itself can be performed, this is done by using the mvdXML standard, building code validation files have been created earlier in this format, and the open source mvdXMLChecker tool.

\(^{53}\) “buildingSMART alliance Common Building Information ...” 2013. 17 Feb. 2015
<http://www.nibs.org/?page=bsa_commonbimfiles>
is used to perform these validations. The result of the validations is then inspected visually by the building owner and feedback is returned to the architect and MEP designer if necessary. This process will need to be repeated multiple times without extra significant effort, and across multiple projects where the stakeholders might be different or other pre- or post-processing operations might be necessary.

This workflow is visualized below:
9.2. Framework implementation

When implementing this with the dataflow framework 17 nodes are needed, of which 7 nodes are input string nodes, and 10 of which contain logic.

First the username, password and url of the two model servers concerned are specified. Then a GetBIMserver call is done to both BIMservers to obtain a reference to the session to both. This references is passed on to the GetProject node that allows the user to select the relevant project, this is then sent to the GetRevision node that allows the user to select the relevant revision to be used, the default is the latest revision.

A BimQL query is then done, in this example first all the IfcDoors are queried from the model on the first BIMserver, then the IfcSlabs are queried from the model on the second BIMserver. These results go into a MergeModel node, so that the two submodels are merged on the first BIMserver. Finally the merged models goes into the mvdXML node along with a URL to the mvdXML validation file. In this case the validation file checks for the presence of a FireProof property on all doors, in this particular case this property is missing from the front door, so it is rendered in red in the 3d panel.
Compared to doing this manually, this process is a lot more efficient and flexible. First of all the nodes can be changed at any moment and the dataflow will reprocess the part that has been changed. Second when looking at the manual process the steps might look as following

1. Log in to BIMserver 1
2. Navigate to the relevant revision
3. Download the revision as IFC
4. Log in to BIMserver 2
5. Navigate to the relevant revision
6. Download the revision as IFC
7. Log in to BIMserver 1
8. Create a working project and two sub projects
9. Upload both revisions to the sub projects
10. Download the merged model as IFC
11. Use the mvdXMLChecker Java tool
12. Upload the resulting file to the working project on BIMserver 1
13. View the results in 3d
14. Repeat above if changes are required
This process has to be repeated every time the workflow needs to be executed again. The process might be written down in a document, but it is not formalized in a standardized way. Therefore the process can not be easily repeated across different projects and needs to be modified as a whole when a new project has different needs. Furthermore because the process is executed manually it is prone to user errors. Finally executed this process manually is time consuming compared to using an automated process, the main areas where time can be saved are:

- No need to manually log in and navigate to separate BIMservers, this is all done from within a single dataflow environment
- No need to manually upload or download files to and from a local drive, which can lead to confusion with file names, versions, etc
- No need to create a separate working project for doing merges across BIMservers
- No need to manually use or install command line or other local useful tools such as the mvdXMLChecker tool
- No need to re-upload the results of these tools to the BIMserver
- No need to repeat any of these steps after the dataflow has been configured once, when revisions, projects or other parameters change the dataflow can be modified and will automatically execute the changed parts.

Having a formal, graphical and flexible representation of the process that can be communicated with other stakeholders and that can be re-used across projects is a significant improvement over the current manual approach or over creating a custom application for this process.
10. Performance

10.1. Introduction

Dataflow programming can lead to improved performance of an application as a whole when comparing to an application created using traditional programming approaches. The main cause for this is not that dataflow applications are inherently faster, but that it is much easier to design concurrent / asynchronous applications that make use of multiple servers or threads to run a computation. Designing concurrent application is often hard when using a traditional programming approach.

When using dataflow programming it is often much easier to see which computations can be performed in parallel. Because the input and output of an operation has been defined and because an operation is free of side effects the dependencies between operations are clear before running the program.

For example in the above computation we can see that the “+” and “/” operations are independent from each other and can therefore be performed at the same time. Therefore the computation can be run in 2 steps instead of the 3 steps used in the imperative version. This optimization is more difficult to see and implement in the imperative version.

This is especially important when operations are performed on a remote server where
latency is a factor, or when operations can be long running, such as with complicated query operations are mvdXML validations.

The second advantage in terms of performance when using dataflow programming is that when a computation changes, or a certain input value changes only the part of the computation that is dependent on this has to update.

The setup when doing these tests was as follows:

<table>
<thead>
<tr>
<th>Computer</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Macbook Pro</td>
</tr>
<tr>
<td>CPU</td>
<td>2,7 Ghz Intel Core i7 (4 cores)</td>
</tr>
<tr>
<td>Total memory</td>
<td>16 GB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Servers</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>BIMserver 1 (port 8082)</td>
<td>8 GB max heap</td>
</tr>
<tr>
<td>BIMserver 2 (port 8083)</td>
<td>8 GB max heap</td>
</tr>
<tr>
<td>Express server (port 3000)</td>
<td>-</td>
</tr>
<tr>
<td>mvdXMLChecker server (port 3001)</td>
<td>-</td>
</tr>
<tr>
<td>MongoDB</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test Model</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>simple house.ifc</td>
<td>122kb</td>
</tr>
</tbody>
</table>
10.2. Network timings and data transfer

Fig. Network timings for example scenario
When running the example scenario we can observe the timings above. Described are the network timings, this excludes time used for rendering the user interface or the 3d preview. The client is set up using web workers so all the network requests run in a separate worker so that their timings are independent from network requests. Network timings are therefore the most accurate way to describe the performance of the dataflow framework.

The total running time of the dataflow graph used in this environment is measured as the time that is needed for all requests to finish, some extra time is needed for actually rendering the 3d preview, which is left out in this case. The total running time is 33.3 seconds. Most time is spent doing the Merge (8.7s), the mvdXML validation (4.7s) and downloading the model for visualization in various steps (3.5s, 3.3s, 3.2s, 3.1s).

In this example all the servers ran on the same machine, so latency and transfer times are excluded, therefore the time is expected to be slower when the applications is running over multiple remote servers. In that scenario however the individual servers have more resources, which can increase response times.

The amount of data transferred is 1317 KB downloaded and 233 KB uploaded. The source model is 122 KB. The model data is transferred as base64 encoded IFC uploaded or base64 encoded Collada data, the Bimsie API expects base64 encoded IFC data, and the ThreeJS based renderer expects Collada files. Most data is transferred when downloading the model for 3d preview rendering, however because many temporary revisions are needed the getAllRevisionsOfProject for the internal working project can also be quite large (224 KB, 217 KB).
10.3. Performance Improvements

The application created for the example scenario can be characterized as a proof of concept. It is not intended for real world usage and large model sizes but merely a study in the possibilities of using the dataflow approach in the AEC domain.

Important to note is that in this example scenario the requests are done sequentially, this leads to especially poor performance because as explained earlier the dataflow approach lends itself well to parallelisation. The optimization depends heavily on the dataflow graph that is used. In the example scenario both of the query paths for the two model servers can be executed in parallel because they are independent. This leads to a drop in running time from 33.3s to 30.1s, a 10% improvement. This is because most of the heavy calculations, the merge and the mvdXML validation, are running in a sequential path.

Two structural improvements can be made if some adaptations are done to the BIMserver API. First the checkin operation or one of its progress helpers could be modified so that it returns the revision that is being created, this saves a getAllRevisionsOfProject call that needs to be done to obtain the revision that was just created. A second improvement would be that query results are also saved on the model server, in the current API it is only possible to download a serialized version of the query result. If further processing is needed on the query results, for example doing a merge, then it needs to be checked in as a new revision again.

This leads to another problem with the prototype that was created, all the model data currently flows through the dataflow client. This means that in the case of very large models they will need to be transferred to the client after every step. In most practical situations the client will have insufficient memory and bandwidth to handle this load.

Finally some existing optimization methods can be applied, namely caching and compression. Caching can be used to save the result of an operation locally and if the input data does not change the result can be re-used. This can even be applied when the dataflow changes, for example if a specific query is executed multiple times against the same model it can be loaded out of a local cache instead of executing the operation again. Compression can be used to decrease the amount of data that is transferred. Many compression techniques exist, some relying on domain knowledge and others
being more generic text based compression methods such as GZIP. Because both IFC and Collada are text based file formats it can be expected that GZIP compression will yield a significant decrease in data size.
11. Conclusion and future work

11.1. Conclusion

In this research a novel framework for creating formal, graphical and reusable representations of the processes that require the use of BIM services is presented. The framework provides a concrete implementation of the workflow steps so that the workflows can be executed automatically, reducing the amount of manual steps involved.

The key characteristics of the framework are usability, extensibility and being open. The usability of the framework by building professionals is achieved by providing an easy to use graphical user interface to compose workflows, the different skill levels of users are considered in different ways to use the framework. The framework is extensible by being open source and by providing APIs and methods to create custom nodes and subgraphs that can be composed into larger workflows. By basing the framework on open source tools and using open standards for data exchange such as IFC the framework can be used and re-used by a broad range of building professionals.

An example scenario has been created to validate the framework where a standard workflow of a building owner is modelled using the nodes that have been developed for this purpose. A sequence of more than 14 manual steps that are error prone and have to be repeated every design iteration can be modelled using 10 logical nodes that can be re-used across projects and design iterations.

The concept of applying dataflow programming principles to provide concrete implementations of the workflows of building professionals has been validated and is a promising avenue for further research into increasing communication and efficiency in the building design and construction process.

11.2. Future work

In terms of performance there are many areas to improve the current proof of concept implementation of the framework. There are also more theoretical issues to performance that require further research to ensure efficient implementation of the dataflow programming environment in the building industry. Several API
improvements as well as directions of research have been proposed in the Performance chapter.

The framework is currently based on the Bimsie APIs for connecting BIM web services. Some further generalizations can be applied to the framework to make it applicable for a broader range of web services without requiring data wrappers. For example the Model datatype could be generalized to be able to refer to multiple types of repositories containing building information models, such as a filesystem, or cloud based collaboration platform.

The set of nodes that has been provided in this research is a limited set that serves as a proof of concept. As further research the toolkit should be expanded to contain a broader range of web services such as energy modelling, rendering, clash detection and issues tracking services. This will increase the applicability of the framework to a broader range of uses.

The BimQL project has been used to create queries on building models. A direction of research might be to explore query language composition by using dataflow nodes instead of the text based approach that is used now. Existing research has been done towards this, for example in the ETL domain to performance querying on large volumes of data.

Finally to increase collaboration when using dataflow graphs research is required into the potential for dataflow graphs to be exchanged using a cloud based platform, so that graphs can be easily shared within the larger building community through a public web service. To further increase collaboration research into the versioning of dataflow graphs is necessary so that changes in graphs can be saved and different versions of a graph can be compared.