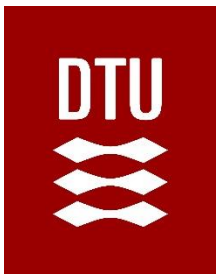
TECHNICAL UNIVERSITY OF DENMARK



PROTOTYPING – Demo Project Documentation

APPENDIX 3

Thesis:	Unlocking the value of Linked Building Data (LBD) - A Lean and integrated management process of temporary construction items (TCIs)
Chapter:	Chapter 6.1 – Demo Project (Input for RQ3)
Method:	Prototyping (Martin & Hanington, 2012)
Purpose:	This demo documentation will serve as a protocol for the prototyping process of the proposed theoretical solution in the thesis. The documentation is structured to follow the consecutive steps of prototyping and spans from the creation of the demo-project-specific BIM data, over the data extraction and translation into an RDF-based knowledge graph to the data processing and to receive the required results.
Author:	Alexander Schlachter
Student number:	182781

Table of Content

List of Figures

List of Tables

Chapter 1. Introduction

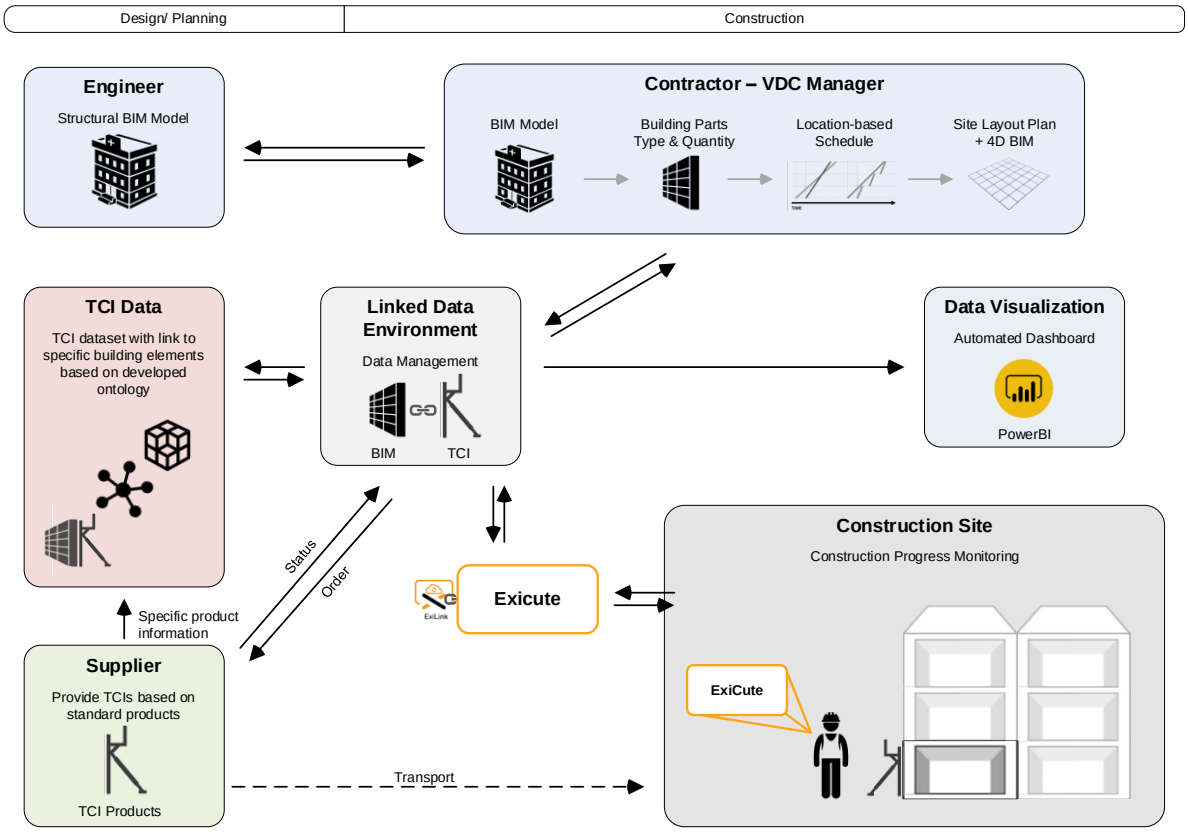


Figure 1: Proposed framework for the innovative management process of TCIs

The Demo Project is part of the prototyping methodology and is used to validate the proof of concept of the proposed solution with a small-scale and fictive building project. The same workflow and the same tools are used as in a real project to create all required data and receive the required output data. As a result of the prototyping process, a functioning system shall be developed that is able to improve the logistic management process of TCIs in a construction project, visualized in figure 1. Recapping the statements in chapter 5.2.1, the overall goal of implementing the proposed solution is to: (a) automatically evaluate the building model geometry, (b) identify required TCIs to each building object of the building model by applying the rule-based algorithm, (c) link the building objects with their respective TCI-information to the building locations and schedule, (d) develop a TCI-utilization plan based on the building objects, their locations, and schedule, (e) enable a passive monitoring of the TCI items with progress tracking data and (f) visualize data automatically and interactively to all relevant stakeholders.

To realize this goal, an integrated system must be established that can receive all required datasets, bring the data in the required and structured format and in the end link the datasets to one combined data graph where the data can be processed in order to receive the needed results. The methodology for creating this integrated system is based on the semantic web and linked data approach that was introduced in chapter 5.2.2. Linked data is enabled by the data model RDF (Resource Description Framework). RDF data is stored in a data graph that consists of triples which are linked with relations and thus, creating an interconnected data network, a graph. Triples always comprise a subject, a predicate, and an object. Predicates have the power to create classes and relations within a data model in order to create a data graph. Using the semantic query language SPARQL allows to retrieve and process the data of the data graph, which is stored in RDF format.

Hence, the output data of the processed data in this project shall hold all information to create a TCI-utilization plan and by that manage and monitor these items throughout the construction project. Data visualization and distribution is considered the last step of the prototyping method and will provide the processed data in a user-friendly and visualized format, distributed to all relevant stakeholder groups. Figure 2 is summarizing the system architecture of the demo project and gives an overview of the consecutive prototyping steps to fulfill goals a) to f), identified earlier.

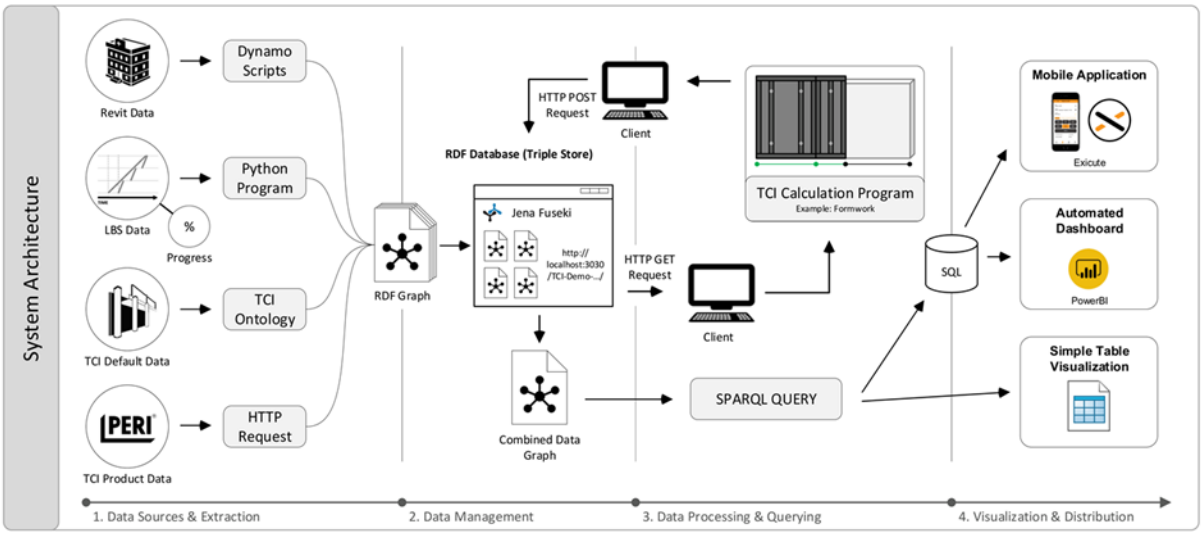


Figure 2: System Architecture of the proposed solution

Chapter 2. Data Sources & Extraction

2.1 Introduction to all data sources

For the demo project, a simple BIM project with a building model and schedule information is created. Walls are selected as a representative of building objects and formwork will represent the TCI-objects. Regarding the creation of the 3D model data and schedule data, common applications in the building industry are chosen. Autodesk Revit is used as the modeling software and VICO office as the complementary software to create a location-based schedule, based on the building elements of the 3D-Model. This schedule information is extended by a progress monitoring of the construction activity through the application Exicute (https://exicute.net/time-management/exicute-app/). The TCI data is generated by creating a new ontology that describes a default formwork set for the purpose of this demo project. The essential information which is needed from each data source is visualized in the following figure.

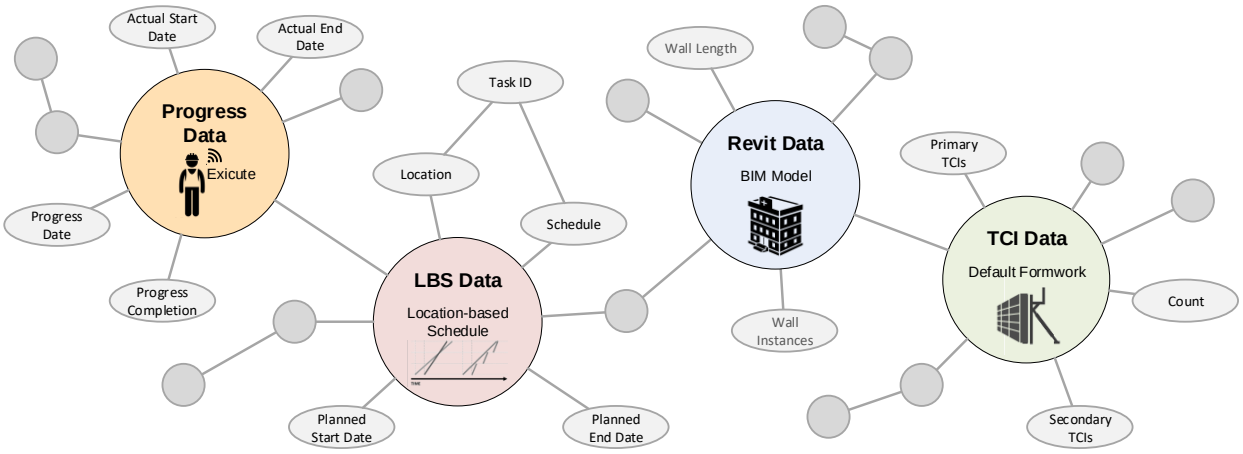


Figure 3: Data Sources with desired parameters

All four datasets are generated from different software applications and in different file formats. However, in the end, the data needs to be integrated into one knowledge graph and linked with previously created relations in the RDF triples. Therefore, the four different datasets need to be first converted into RDF triple files and then uploaded into an RDF triple store to create the respective data graph. Here, the open-source triple store Jena Fuseki is used to upload the different datasets into one database. In this triple store, the four individual datasets can then be integrated into one combined knowledge graph.

By then querying the linked data, the required results can be generated and used to improve the logistics management process of TCIs. This processed data can hence be visualized according to the stakeholders' needs in order to be used in real life.

2.2 3D-Model-Data

The BIM modelling software Revit 2019 is used to create a simple building model for the purpose of this prototyping. The model file name is "AS_Demo_Project" and contains a total of four levels (Foundation = Level 1, Ground Floor = Level 2, First Floor = Level 3, Attic = Level 4), a total of 14 walls, two floor slabs, and a roof. Figure 4 shows the 3D-view of the demo project.

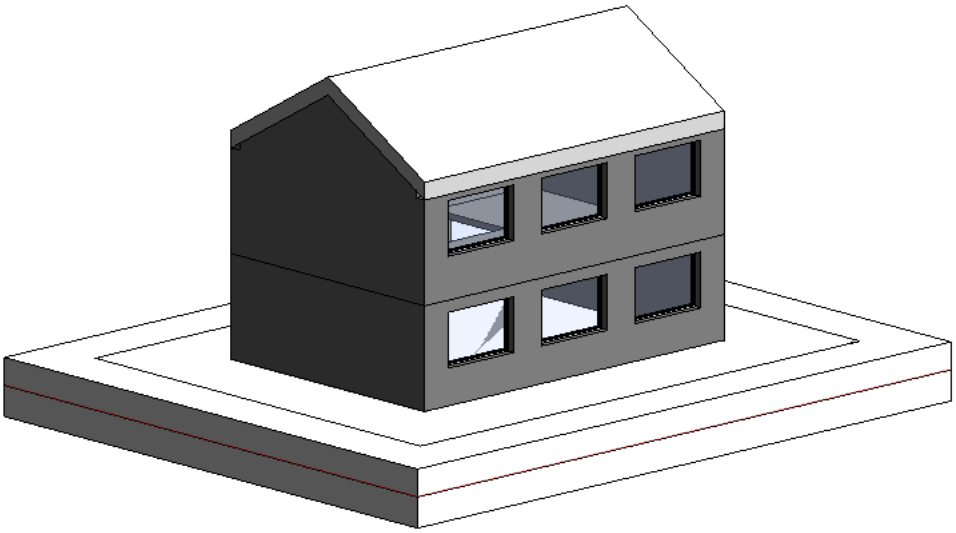


Figure 4: 3D-Building Model visualization from Revit

The following table gives an overview of the building objects parameters in the demo project.

No.	Assembly Description	Family and Type	Element ID	Instance Code	Or.	Lev.	Area m²
	Concrete - insitu						
	Floor: Foundation - 900mm (Cast-in-Place)						
1	Slab on Grade	Floor: Foundation - 900mm …	586351	(13)21.40,01.0		1	140
	Floor: Generic - 200mm (Interior) 2						
1	Floor Construction	Floor: Generic - 200mm (Interior)	595994	(23)23.06,02.2		2	96
	Basic Wall: Concrete - 400mm (Cast-in-Place)						
1	Exterior Walls	Basic Wall: Concrete - 400mm …	586063	(12)11.15,05.1.N2	N	1	19.2
2	Exterior Walls	Basic Wall: Concrete - 400mm …	645092	(12)11.15,05.1.N1	N	1	19.2
3	Exterior Walls	Basic Wall: Concrete - 400mm …	585968	(12)11.15,05.1.E	E	1	24.0
4	Exterior Walls	Basic Wall: Concrete - 400mm …	644734	(12)11.15,05.1.S2	S	1	19.2

No.	Assembly Description	Family and Type	Element ID	Instance Code	Or.	Lev.	Area m²
5	Exterior Walls	Basic Wall: Concrete - 400mm …	585914	(12)11.15,05.1.S1	S	1	19.2
6	Exterior Walls	Basic Wall: Concrete - 400mm …	640260	(12)11.15,05.1.W	W	1	24.0
7	Exterior Walls	Basic Wall: Concrete - 400mm …	642905	(12)11.15,05.2.N2	N	2	19.2
8	Exterior Walls	Basic Wall: Concrete - 400mm …	644964	(12)11.15,05.2.N1	N	2	19.2
9	Exterior Walls	Basic Wall: Concrete - 400mm …	641019	(12)11.15,05.2.E	E	2	24
10	Exterior Walls	Basic Wall: Concrete - 400mm …	640605	(12)11.15,05.2.S2	S	2	19.2
11	Exterior Walls	Basic Wall: Concrete - 400mm …	643638	(12)11.15,05.2.S1	S	2	19.2
12	Exterior Walls	Basic Wall: Concrete - 400mm …	640666	(12)11.15,05.2.W	W	2	24.0
13	Exterior Walls	Basic Wall: Concrete - 400mm …	641779	(12)11.15,05.3.E	E	3	11.2
14	Exterior Walls	Basic Wall: Concrete - 400mm …	636179	(12)11.15,05.3.W	W	3	11.2
	Roof						
	Basic Roof: Generic - 500 mm						
1	Roofing	Basic Roof: Generic - 500 mm	586094	(27)19.10,08.3		3	130

Table 1: Resource list of demo project building model

The goal is to link information about the schedule, the progress, and the required temporary construction items to the building object data received from Revit. Hence, a direct relation of TCIs requirements to each instance object in the building model is created, including all required data to passively monitor TCIs. As the building object is the central unit for all software applications, Revit will provide the main data source to build the knowledge graph which combines all data sources by linking them together. In order to make the data usable for the proposed solution, the model has to be parsed into structured data that describe the required model information. Therefore, different data has to be extracted from the Revit file that thoroughly describes the building objects. In terms of the prototyping, only wall objects are considered to fulfill the requirements of concept proofing.

This is being done with Dynamo scripts, directly in Revit, which automatically generate RDF triples from the available model data. The scripts are based on already existing solutions in GitHub (https://github.com/MadsHolten/OPM-REST/tree/master/tools/dynamo-scripts). The advantage of using visual scripting with Dynamo is that the data is directly converted in the right data format which can then be uploaded to an RDF triple store (RDF database) as Jena Fuseki. The following list is presenting all information that is extracted from Revit via the Dynamo scripts.

Revit Elements	Revit property	OWL property	OWL object	Datatype
Wall, Level	Revit GUID (UniqueId)	props:Revit_GUID	e.g. "450d31df-4383-4692-9be4-9c0935e083ef-0008f0ba"	xsd:string
Wall, Floor	Element ID	Props:Element_ID	e.g. "585914"	xsd:string
Wall, Floor	Inst. Element	rdf:label	e.g. "(12)11.15,05.1.S"	xsd:string
Wall, Floor	Inst. Element	rdf:type (a)	bot:Element	URI
Wall, Floor	Inst. Element	rdf:type (a)	product:Wall, product:Floor	URI
Wall, Floor	Inst. Element	rdf:type (a)	ont:Concrete400MmCastInPlace	URI
Wall, Floor	Type	rdf:label	e.g. "Basic Wall: Concrete - 400 mm (Cast-in-Place)"	xsd:string
Walls, Floors	Type	rdf:type (a) rdfs:SubclassOf	owl:Class; product:Wall	URI URI
Wall, Floor	Width Type Parameter	props:thickness	e.g. "0.4 m"	xsd:decimal
Wall, Floors	Material Type Parameter	props:material	e.g. "Concrete, Cast-in-Place, Grey"	xsd:string
Wall, Floor	Area Inst. Parameter	props:area	e.g. "27.0 m²"	xsd:decimal
Wall	Length Inst. Parameter	props:length	e.g. "8.4 m"	xsd:decimal
Wall	Height Inst. Parameter	props:height	e.g. "3.0 m"	xsd:decimal
Level	Inst. Element	rdf:type (a)	bot:Storey	URI
Level	Inst. Element	rdf:label	e.g. "Level 1"	xsd:string
Level	Base Constraint Level	bot:hasElement	inst:wall; inst:floor	xsd:string
Wall, Floor	Surrounding Element	bot:adjacentElement	inst:wall	URI
Wall	Wall Orientation	props:angle	e.g. "90.0"	xsd:decimal

Table 2: Properties exported from Revit

With the generated Revit data graph, all required information from the building model is exported and ready to create relations with the other data graphs, specified earlier.

The first approach with Dynamo aimed to create RDF triple files for each property that needs to be exported from Revit. By doing this, the model data was saved as turtle files (.ttl) and then uploaded into the RDF triples store in Jena Fuseki. Figure 5 shows the Jena Fuseki interface to upload the generated turtle files into the triple store.

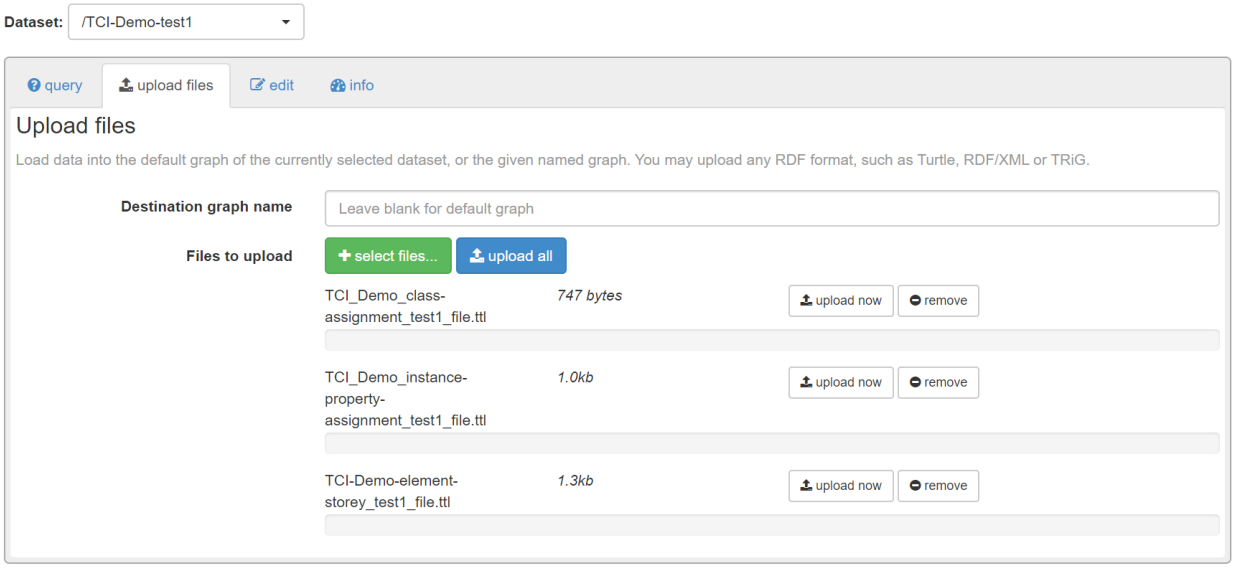


Figure 5: Manual approach of uploading the required data files into the triple store

The fact that this approach required a manual upload of all property data files into the triple store makes it very time-consuming and limits the automation factor. In order to increase automation, the next approach aimed to directly write the triples into the triple store. Therefore, the existing scripts are extended with a Python code which enables to update a data set in the triple store with the extracted model data. By sending an HTTP POST request to the Fuseki database, the generated triples can be written into the specified dataset by specifying its URL. In Fuseki, there are two options for inserting data to an existing dataset. http://localhost:3030/TCI-Demo-Revit-data/data is the URL where data is stored as an RDF-graph. It can be also be used as the URL for the HTTP POST request as the Fuseki database allows to directly write to the storage location. Another, slightly more difficult approach is to write the HTTP POST request to http://localhost:3030/TCI-Demo-Revit-data/update. In theory, this will update the dataset with new data, but in practice, it has the same result as the other method. The only difference is that with the second approach, the triples are written with their namespaces as prefixes are not published.

Figure 6 shows the resulting data graph that combines the different turtle files in the triple store, based on their relation.

```
wallinst:450d31df-4383-4692-9be4-9c0935e083ef-0008f0ba
        a                   product:Wall , ont:Concrete400MmCastInPlace ;
        rdf:label           "(12)11.15,05.1.S1" ;
        bot:adjacentElement wallinst:40cab1d1-1d6f-47a3-9afb-bd8c6300ff7e-0009c504 , wallinst:c1037085-1aff-4644-8770-66dc41edbf0b-0009d67e ;
        props:Element_ID    "585914" ;
        props:Revit_GUID    "450d31df-4383-4692-9be4-9c0935e083ef-0008f0ba" ;
        props:angle         0.0 ;
        props:area          19.2 ;
        props:height        3.0 ;
        props:length        6.2 ;
        props:level_simple  "Level1" .

wallinst:450d31df-4383-4692-9be4-9c0935e083ef-0008f0f0
        a                   product:Wall , ont:Concrete400MmCastInPlace ;
        rdf:label           "(12)11.15,05.1.E" ;
        bot:adjacentElement wallinst:450d31df-4383-4692-9be4-9c0935e083ef-0008f14f , wallinst:c1037085-1aff-4644-8770-66dc41edbf0b-0009d67e ;
        props:Element_ID    "585968" ;
        props:Revit_GUID    "450d31df-4383-4692-9be4-9c0935e083ef-0008f0f0" ;
        props:angle         90.0 ;
        props:area          24.0 ;
        props:height        3.0 ;
        props:length        8.4 ;
        props:level_simple  "Level1" .

wallinst:450d31df-4383-4692-9be4-9c0935e083ef-0008f14f
        a                   product:Wall , ont:Concrete400MmCastInPlace ;
        rdf:label           "(12)11.15,05.1.N2" ;
        bot:adjacentElement wallinst:c1037085-1aff-4644-8770-66dc41edbf0b-0009d7e4 , wallinst:450d31df-4383-4692-9be4-9c0935e083ef-0008f0f0 ;
        props:Element_ID    "586063" ;
        props:Revit_GUID    "450d31df-4383-4692-9be4-9c0935e083ef-0008f14f" ;
        props:angle         180.0 ;
        props:area          19.2 ;
        props:height        3.0 ;
        props:length        6.2 ;
        props:level_simple  "Level1" .
```

Figure 6: Excerpt of the resulting Revit data graph showing wall instances

The dataset of the 3D-Model is built on existing ontologies, especially from the Linked Building Data community group, describing the context of a building in Linked Data format. Thus, there was no need to develop a new ontology for this dataset. The entire Revit data graph and all following data graphs of this demo project can be found in the GitHub repository LBD-for-TCI (https://github.com/Alex-Schlachter27/LBD-for-TCI/tree/master).

2.3 LBS Data

Schedule information is crucial data in the process of planning and monitoring items in a construction project as it provides information about when specific building objects are planned to be executed. In the construction industry, this is traditionally done with Gantt-Charts, which represent construction activities that include information about when a task is planned to start, how long it will take, and when it will end. A further development of this approach is the Location-Based Scheduling (LBS) which extends the time information with the location dimension. Besides many proven advantages, this scheduling approach allows a continuous and efficient construction process by reducing waste.

In this demo project, the LBS-approach is utilized with the help of VICO Office, a scheduling software that enables a BIM-integrated and location-based scheduling process (figure 7). The advantage of location-based scheduling is that the location dimension provides a new level of detail that enables to sequence the construction progress based on both the sequence logic of construction and the logic of location to avoid interferences and waste. For the passive planning of formwork, this approach furthermore allows quantifying the amount and types of formwork which is required for the individual tasks. Each task is based on a schedule sequence and its specific location and therefore, the construction management would know exactly when and where which kind and amount of formwork is needed on site.

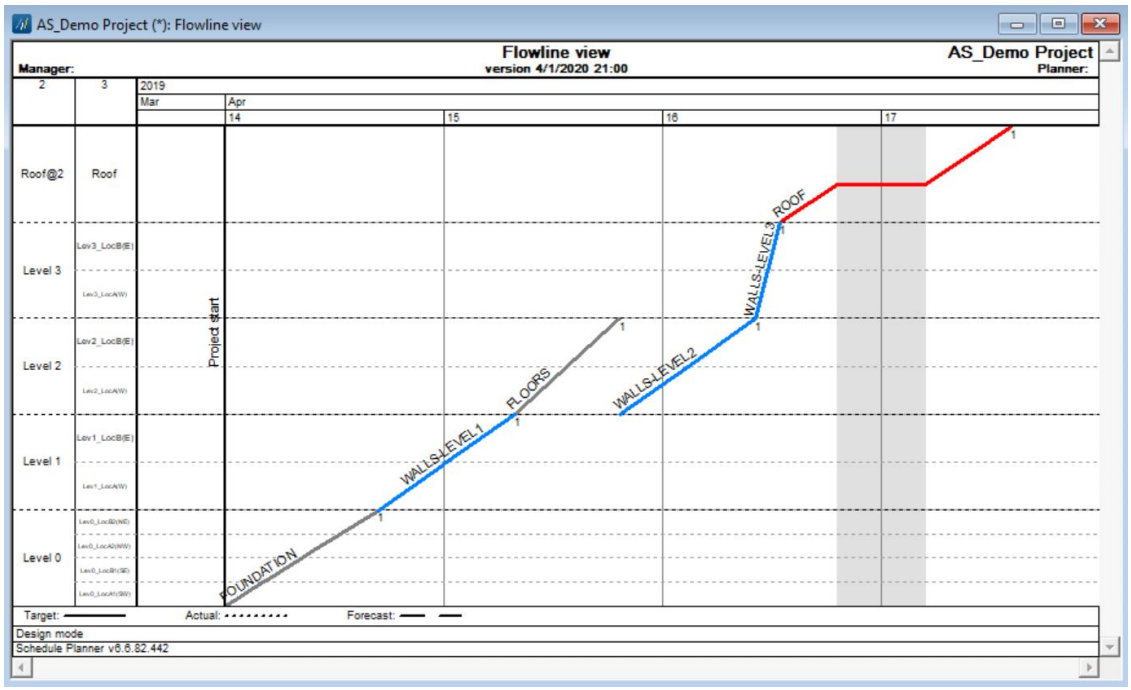


Figure 7: Location-based schedule of the demo project

After establishing the schedule with tasks that are linked to each of the available building objects and their location in the model, all available data has to be extracted in order to explore the data for the valuable instances. In this case, VICO provides the data in five different datasets (Cost Plan, LBS, Scheduling System, Takeoff System, and CAD Model System) as JSON files that are interconnected with common ID codes. Currently, Exigo is exporting only three (Cost Plan, LBS, Scheduling System) of the five available datasets for the use of the progress monitoring application Exicute. The issue is explained in the following table, that presents the required data from VICO, how it can be translated into RDF, and in which dataset it can be found.

VICO property	VICO dataset	OWL property	Datatype
Element ID	CAD Model System	props:Element_ID	xsd:string
compLoid	All systems	tci:hasCompLoid	xsd:string
locLoid	LBS	tci:haslocLoid	xsd:string
Location Name	LBS	tci:hasLocation	xsd:string
schedloid	Scheduling System	tci:hasschedLoid	xsd:string
taskloid	Scheduling System	tci:hastaskLoid	xsd:string
taskPlannedStartDate	Scheduling System	tci:taskPlannedStartDate	xsd:DateTime
taskPlannedEndDate	Scheduling System	tci:taskPlannedEndDate	xsd:DateTime
taskActualStartDate	Scheduling System	tci:taskActualStartDate	xsd:DateTime
taskActualEndDate	Scheduling System	tci:taskActualEndDate	xsd:DateTime
taskProgressDate	Scheduling System	tci:taskProgressDate	xsd:DateTime
taskProgressCompletion	Scheduling System	tci:taskProgressCompletion	Xsd:nonNegativeInteger

Table 3: Required data from VICO and where it is located

Going through the raw data of VICO, it was recognized that all required data regarding the schedule and location is already available through the export of Exigo with the extraction tool ExiLink. However, in order to create a relation between the VICO data and the 3D-Model data, some reference code is needed in order to map the two datasets. In this case, the reference code would be the Element ID, which is created in Revit for every single building object and gets transferred to the CAD Model System dataset in VICO. Besides the CAD Model System, the Takeoff System also needs to be exported in order to link the Element ID to the schedule and location data, because the relation goes through the component ID (compLoid). This, however, is not currently utilized from Exigo and would require processing much more data as it is currently processed. Hence, a solution is needed that exports all of the required data.

In this case, two options are available to solve the problem. The first is to modify the program that was coded from Exigo to extract the VICO data into a SQL format, to also support the two missing datasets. The other option is to write a program that is receiving the required JSON files, selects the data that is needed as specified in table 3 and writes it directly in RDF language to a chosen dataset in a triple store.

For the demo-project, a manual extraction of the data was chosen to simplify the process and get a quick result. For the further development of the solution and the later use of the process, however, a program needs to take care of the data extraction of the VICO data. Either to use the SQL data from the existing VICO-extraction from Exigo with the extraction tool ExiLink or to develop an own program which is able to convert the raw JSON data into RDF triples and write it to a defined triple store.

In order to store the data in Linked Data terminology, first, there is a need to develop a new ontology that describes the data from VICO (see OWL property column in table 3) and is able to create a link to the other datasets through property relations. This LBS ontology was built with an open-source tool called "Protégé" and has the following fictive namespace: http://test/lbs/. The next step in processing the data into RDF format is to map the existing VICO data within the developed ontology classes, instances, and properties. The resulting data graph for the LBS data is shown below.

```
inst:1000.0.145882  a                lbs:CompLoid , product:Wall ;
        lbs:hasCompLoid              "1000.0.145882" ;
        lbs:hasLocation              "Lev1_loca(w)" ;
        lbs:haslocLoid               "1000.0.355001" ;
        lbs:hasschedLoid             "1000.0.321768" ;
        lbs:hastaskLoid              "1000.0.358588" ;
        lbs:taskActualEndDate        "NULL"^^xsd:dateTime ;
        lbs:taskActualStartDate      "NULL"^^xsd:dateTime ;
        lbs:taskPlannedEndDate       "2019-04-08 07:28:48.000"^^xsd:dateTime ;
        lbs:taskPlannedStartDate     "2019-04-04 11:00:00.000"^^xsd:dateTime ;
        lbs:taskProgressCompletion   "0.0"^^xsd:nonNegativeInteger ;
        lbs:taskProgressDate         "NULL"^^xsd:dateTime ;
        props:Element_ID             "585914" ;
        props:Revit_GUID             "450d31df-4383-4692-9be4-9c0935e083ef-0008f0ba" .

inst:1000.0.145920  a                lbs:CompLoid , product:Wall ;
        lbs:hasCompLoid              "1000.0.145920" ;
        lbs:hasLocation              "Lev1_locb(e)" ;
        lbs:haslocLoid               "1000.0.355015" ;
        lbs:hasschedLoid             "1000.0.321768" ;
        lbs:hastaskLoid              "1000.0.358593" ;
        lbs:taskActualEndDate        "NULL"^^xsd:dateTime ;
        lbs:taskActualStartDate      "NULL"^^xsd:dateTime ;
        lbs:taskPlannedEndDate       "2019-04-09 11:57:36.000"^^xsd:dateTime ;
        lbs:taskPlannedStartDate     "2019-04-08 07:28:48.000"^^xsd:dateTime ;
        lbs:taskProgressCompletion   "0.0"^^xsd:nonNegativeInteger ;
        lbs:taskProgressDate         "NULL"^^xsd:dateTime ;
        props:Element_ID             "585968" ;
        props:Revit_GUID             "450d31df-4383-4692-9be4-9c0935e083ef-0008f0f0" .
```

Figure 8: Excerpt of the resulting LBS data graph showing wall instances

2.4 Progress Monitoring Data

As seen in figure 8, some of the objects of each wall instance are returning "NULL" as their value, meaning that their value has not yet been specified. This information is classified as progress data because it provides information about the current progress of the construction activities and is an optional feature of the VICO schedule planner to monitor the construction activities based on the planned schedule. In Exigo, a cloud platform was developed to generate this information from the construction site and write it back to the schedule.

The platform is called Exicute and includes an application that allows the contractors on a construction site to document their progress within a mobile app. The cloud platform then receives the information and updates the VICO data with information as "Progress Date", specifying the date when the progress is documented, "Progress Completion" in % and per task as well as the "Actual Start Date" and Actual End Date" of each construction task. Figure 8 visualizes the data system and the workflow of the Exicute platform.

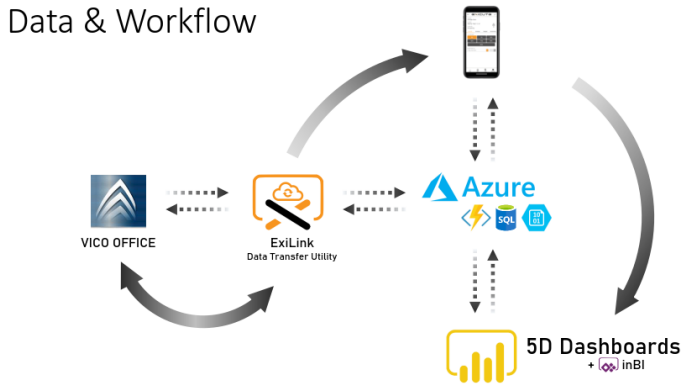


Figure 9: Data system and workflow of Exicute

For the purpose of the proposed solution in this thesis, the already existing progress monitoring of Exicute can be utilized to establish passive monitoring of TCIs, based on the progress of the permanent building parts as there will be a direct relation between each building element and the temporary construction items, required to construct the element. Hence, the TCI utilization plan with its information about when and where, which type and quantity of TCIs are needed on site, gets updated with every progress update from the construction site. This passive monitoring of TCIs already significantly improves the management of these items. However, further development might include IoT-tracking sensors on TCIs to not only receive the information where the items are supposed to be but also create the ability to verify their actual location. This idea is further explained in chapter 6.3 of this thesis.

In conclusion, the progress monitoring data can be seen as a data extension of the LBS data and thus, will be managed within the LBS dataset.

2.5 TCI-Data

This dataset comprises the main knowledge graph of the TCI-Demo-project as it represents the Temporary Construction Items in a newly established TCI Ontology (http://test/tci/). The individuals of the ontology are forming the default set of formwork, which is used to calculate the TCI-utilization plan before the supplier and specific TCI-products are known. The default formwork set is developed with consideration of existing formwork solutions. The following figure is presenting the default formwork set with their specific dimensions.

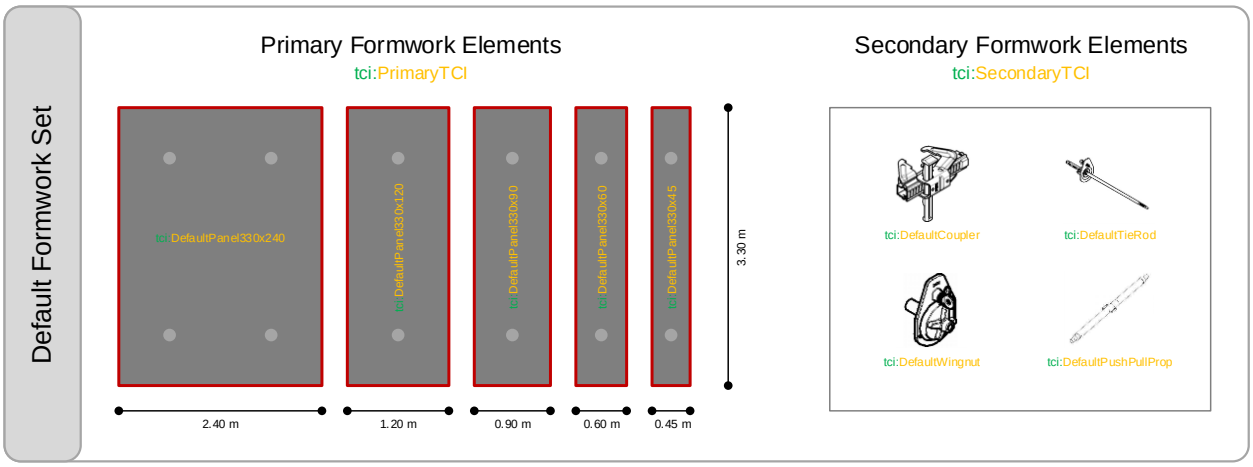


Figure 10: Default Formwork Set

Based on the individual standard dimensions, the quantities of TCIs can then be calculated for a specific wall layout. For simplification reasons, a premise is defined for the creation of building models. This premise requires that the BIM model and schedule are aligned with the construction process. This means that each building element can be assigned to one unique construction task, consisting of schedule and location information, in order to avoid splitting up building elements when creating the location-based schedule. The length of a wall, for example, represents the section of a wall that is constructed in one sequence with a closed set of formwork (a formwork cycle). Hence, the formwork quantity is calculated for each wall section and according to the sequence of construction. In this regard, construction benefits from location-based scheduling as it allows to schedule a sequence of tasks based on their location and following a specific logic. Having the parameters of a wall as length and height consequently enables to quantify the required forms with standard dimensions. Thus, the stated premise ensures the correct deduction of formwork quantities.

Next to the default formwork set, which already gives precise information about what quantities of TCIs are required over time in the construction project, the solution must allow to include specific product catalogues as soon as a supplier and their specific solution is selected for a construction site. By including the actual products which are used on site, the TCI utilization plan directly represents what is going on and what should happen on site and hence, has the same level of detail as a location-based construction schedule would provide for permanent building items as slabs and walls.

In this demo project, the formwork was only calculated with the default set, as it already serves as a proof of concept. Yet, an additional ontology was developed for an example product for the further development of the solution. The chosen example product catalogue is based on the formwork solution "MAXIMO MX15" from the formwork and scaffolding supplier PERI. The complete product catalogue, provided from PERI, can be found with the following URL: https://www.peri.com/brochures/jcr:af4fc9a6-1bca-40ef-ba82-849c803bd562/MAXIMO-Panel-Formwork.pdf. As with the default formwork set, an ontology was created with the information from the provided product catalogue, having the following fictive namespace: http://test/peri/. In this case, the ontology was created manually to represent the product catalogue in RDF language. In the future development of the proposed solution, ontologies of specific product catalogues should be created by the supplier itself which can then be accessed through an open URL. A business case for suppliers could be established that would create incentives to publish their products as RDF data graphs. An example where this future scenario is already put in practice is the BAUKOM /catalogue, where precast building elements are listed in a data model of RDF ontologies (Costa & Pauwels, 2015). In order to differentiate the TCI utilization plan regarding its status, if a specific product catalogue is available, a property status needs to be included in the TCI data. Before a supplier is chosen and a specific product can be selected, the default formwork is classified as *opm:Assumed*. In contrast, specific products will be classified as *opm:Confirmed*. The algorithm that calculates the formwork quantities shall, therefore, look for the property status if a confirmed product is available and if this condition is true, the confirmed product catalogue shall be preferred over the default set of formwork. In this way, it is assured that the TCI utilization plan always considers the level of detail regarding TCIs that are currently available in the construction project.

Chapter 3. Data Management

As presented in the export of the Revit data, the RDF triple store "Apache Jena Fuseki" is utilized to store all the triple relations as data graphs. Fuseki is a SPARQL server, that is able to store RDF data and allows to query the data over HTTP, based on SPARQL 1.1 protocols for query and update. The storage system also provides security by using an authentification system and is accessible through the URL http://localhost:3030/. In general, a triple store is a database to store and query RDF triples. These triples can be stored within the repositories of the triple store. Repositories are databases within an RDF triple store that must have one default data graph and zero or more named graphs.

The data in a triple store is can be processed directly in the interface of Fuseki, using a SPARQL query. For more advanced data processing, the data can be accessed through a SPARQL query over HTTP request on the SPARQL endpoint URL, which allows to receive queried data, process it in a program, and write it back to the triple store. Figure 11 shows a simple configuration of this advanced data processing.

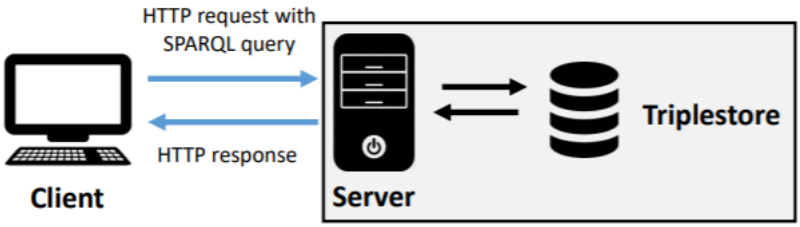


Figure 11: SPARQL query over HTTP request with Triple Store

First of all, the data which is extracted from different software applications of the construction industry is stored in separate datasets. On the one hand, it is simple to write data to an individual dataset as its URL is accessible. On the other hand, having the data in individual datasets allows to receive the data with a simple HTTP request, containing a SELECT query. Due to this flexibility in data management, the data is first stored in individual datasets in the Fuseki triple store.

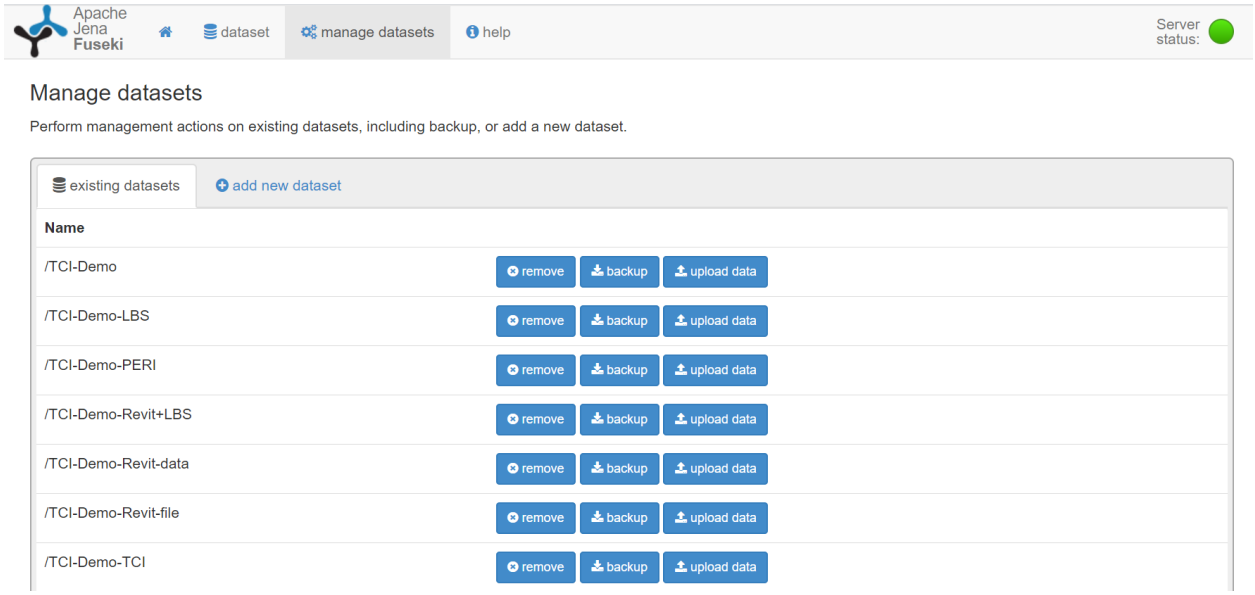


Figure 12: Fuseki Triple Store - Created datasets

The raw data from the previous chapter is stored in the individual datasets with the endings /TCI-Demo-Revit-data, /TCI-Demo-LBS, /TCI-Demo-TCI and /TCI-Demo-PERI (figure 12).

The data graph /TCI-Demo-PERI is containing a specific product catalogue of the formwork solution PERI MAXIMO MX15 and is not used in this Demo Project. The /TCI-Demo dataset is kept empty as this will be used as the combined data graph where the individual datasets are put into named graphs and then are queried together into a combined data graph. Furthermore, the results of the automatic formwork planning also need to be stored in the combined data graph in order to be able to derive the TCI utilization plan from that graph. The use of named graphs allows to query data from all named graphs in one dataset and thus, combining the information. Therefore, the individual data graphs must be stored into one common dataset. This can be done by directly storing the data into named graphs or combining them subsequently with a federated query. In this demo project, however, the data graphs are copied manually to individual named graphs of the dataset /TCI-Demo and are then added into a combined data graph by querying each named graph. How this has been done in detail is explained in the following chapter. The following two queries show how to copy named graphs in order to combine the data.

```
ADD <http://localhost:3030/TCI-Demo/data/Revit> TO GRAPH <http://localhost:3030
/TCI-Demo/data/RevitLBS> ;
ADD DEFAULT TO GRAPH http://localhost:3030/TCI-Demo/data/Combined ;
```

Figure 13: Queries to Add named graphs to another graph

Chapter 4. Data Processing & Querying

In the last step, all data was stored in individual graphs in the fuseki triple store http://localhost:3030/. By that, the required raw data is ready to get processed to develop a TCI utilization plan.

The relation between the LBS and Revit data is already created with the common property "Element ID". With a simple owl:sameAs relation, all wall instances from both data graphs can be mapped in order to have both the building element and time information. Thus, the main challenge of this chapter is the creation of a rule-based algorithm that calculates the TCI requirements for each building element and hence, links the Revit data with the TCI data.

In this demo project, the automatic planning of TCIs with a rule-based algorithm is simplified and serves as a proof of concept for the further development of the solution. The following list defines all simplifications:

a) Formwork is chosen to represent all TCIs that can be linked to a model object

b) One dimensional calculation of vertical formwork for the construction of concrete walls

c) Height of forms is always greater than the wall height, hence no need for consideration of heights

d) Utilization of only a default set of forms with specific parameters

e) Assumption that walls in the building model are modeled as they are constructed, meaning that the wall separation in the model represent the sequencing of the construction activity which has to be covered by formwork

f) Corner Panels are not considered in the demo project

g) Formwork is calculated for the entire length of each modeled wall

The specified simplifications in the demo project enable to focus on the essential aspect of prototyping a solution without restricting the developed solution to be extended again later to cover the whole context. In this case, the created algorithm must possess a certain flexibility to be modified and adjusted. Ceiling formwork, for example, can be considered by adding a dimension to the calculation and the Revit data can already specify corners to also consider corner panels in the demo project.

Other TCIs as scaffolding or fencing might be considered as well by creating a simple calculation and a link to an existing model object. Thus, the chosen simplifications ease the development of a proof of concept but don't add any insurmountable obstacles to the solution development. Furthermore, the simplified demo project still requires some effort to work and is seen as a valid project for the purpose of prototyping.

The problem of placing vertical standard forms on a given wall geometry mainly depends on the length and height of the wall. Given that the height of the walls (h = 3.00 m) of the demo building model is at all times shorter than the height of the default standard forms (h = 3.30 m), the only parameter that impacts the quantities of forms and formwork layout is the length. Based on the given standard lengths of the formwork elements as well as the given length of the walls, an algorithm can calculate the formwork layout with the least amount of forms used. The following rules shall be applied in the algorithm:

- Formwork is calculated for each wall instance in the given Revit dataset
- For the given wall length, the least amount of standard forms shall be calculated
- The default formwork set is based on the standard formwork MAXIMO MX15 and consists of the following primary TCIs: DefaultPanel330x240, DefaultPanel330x120, DefaultPanel330x90, DefaultPanel330x60, DefaultPanel330x45
- If the default formwork layout cannot cover the entire wall, the empty space will be filled with filling material (TimberFilling) which is specified by its length
- Secondary TCIs as couplers and wingnuts are items that are supporting the primary TCIs and are quantified based on rules from existing formwork solutions and the calculated quantity of standard forms
- The following secondary TCIs are used in the demo project: DefaultCoupler, DefaultPushPullProp, DefaultTieRod, DefaultWaler, DefaultWingnut

The intended formwork calculation for an example wall as specified in the ruleset is visualized for a better understanding in the following figure. It shows the required information from both datasets (Revit and TCI) and how the algorithm shall calculate the formwork utilization in order to use the least amount of forms and be able to cover the whole wall length with formwork.

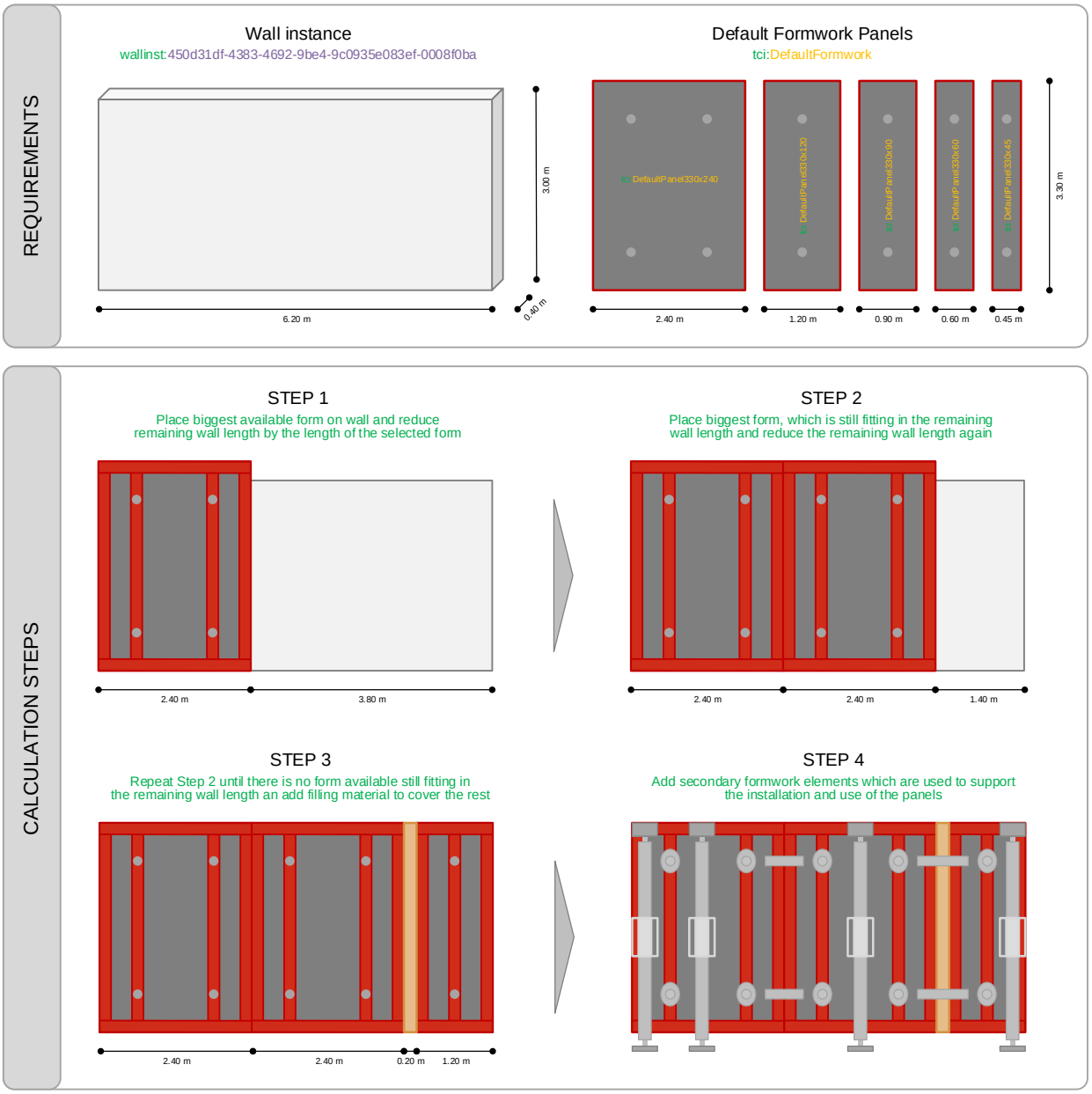
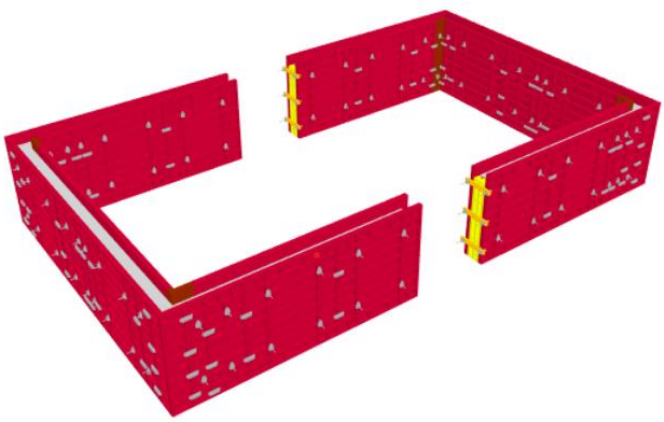


Figure 14: Formwork Calculation Visualization

Integrating these rules into a single calculation required a certain flexibility and calculation power using different datasets. Initially, it was planned to run a SPARQL query directly against the developed datasets in the triple store. However, due to the limited flexibility of such a query, it was decided to write a small program that accesses the data from the triple store via a SPARQL query over an HTTP request to the SPARQL endpoint. After receiving the required data from the triple store, the program calculates the required TCIs for each instance building element and writes the data back to the Fuseki triple store.

As existing code blocks in Javascript (**fuseki.js**) are used to get the data from the triple store with an HTTP-request, Javascript was chosen as the programming language to develop the small program. However, other programming languages as Python are able to do the same. The developed program is named **index.js** and can be found in the GitHub repository LBD-for-TCI under Demo Project (https://github.com/Alex-Schlachter27/LBD-for-TCI/tree/master).

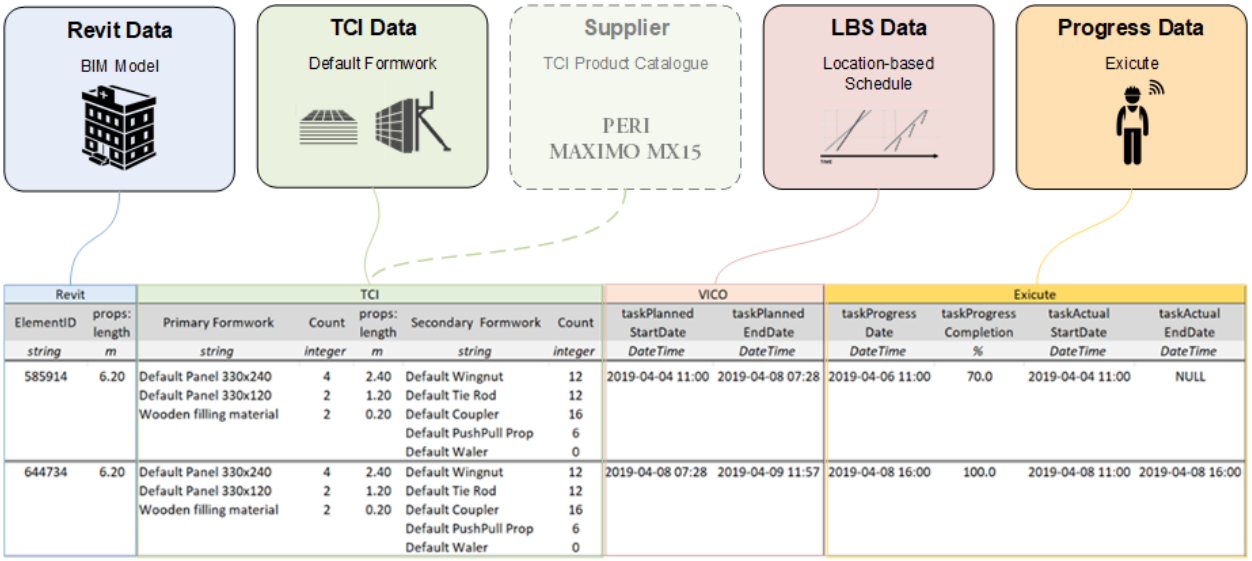
To provide some guidance for the development of the calculating functions in the program, an automatic formwork planning software from PERI (https://quicksolve.peri.app/) is explored. In this application, quantities of formwork from a selected product catalogue (in this case MAXIMO MX15) are calculated by specifying the geometry and sequencing of a wall structure. Figure 15 shows an example formwork sequence and the respective part list consisting of primary TCIs as the standard panels and secondary TCIs.



		1	2	Maximum Pcs.	Correction Pcs.	Sum Pcs.	Weight kg
	ALIGNMENT COUPLER BFD Art.-No. 023500 4.58 kg	12	16	16	0	16	73.28 kg
	WALER 85 Art.-No. 023551 8.52 kg	6	6	6	0	6	51.12 kg
	TRIO BULKHEAD TIE TS, GALV. Art.-No. 023640 1.14 kg	12	12	12	0	12	13.68 kg
	WINGNUT PIVOT PLATE DW15 GALV Art.-No. 030370 1.66 kg	12	12	12	0	12	19.92 kg
	MAXIMO-PANEL MX 270X240 -/- Art.-No. 112006 336.00 kg	4	6	6	0	6	2016.00 kg
	MAXIMO-PANEL MX 270X120 Art.-No. 112022 186.00 kg	2	2	2	0	2	372.00 kg
	MAXIMO-PANEL MX 270X 45 Art.-No. 112078 77.50 kg	2	-	2	0	2	155.00 kg
	WINGNUT MX15 Art.-No. 112386 2.58 kg	12	16	16	0	16	41.28 kg
	TIE MX15 30-40 Art.-No. 112464 4.43 kg	12	16	16	0	16	70.88 kg
	MAXIMO-PANEL MX 270X 30 Art.-No. 112090 61.80 kg	-	2	2	0	2	123.60 kg
	Plywood Width: 40,0 cm, Height: 240,0 cm, Depth: 2,1 cm	2	2	2	0	2	
	Filler timber Width: 8,0 cm, Height: 240,0 cm, Depth: 10,4 cm	6	-	6	0	6	

Figure 15: Wall covered in formwork and the respective part list from PERI Quicksolve

Two reports of example projects in PERI Quicksolve are included in the thesis as appendix 7, helping to develop the prototype solution. By analyzing and exploring the calculating algorithm of the existing solution from PERI, information was gathered to develop the calculation in the prototyping program. In addition to using PERI Quicksolve as a supporting tool to develop the program, a solution-based approach was selected in which the desired solution is first outlined to identify all required data. This approach provides a starting point for a structured development of the program script as the program can be coded step by step to return the desired output. The first step is to define the raw data from different data graphs that are needed for the calculation (figure 16).



Revit		TCI					VICO		Exicute			
ElementID	props: length	Primary Formwork	Count	props: length	Secondary Formwork	Count	taskPlanned StartDate	taskPlanned EndDate	taskProgress Date	taskProgress Completion	taskActual StartDate	taskActual EndDate
string	*m*	*string*	*integer*	*m*	*string*	*integer*	*DateTime*	*DateTime*	*DateTime*	*%*	*DateTime*	*DateTime*
585914	6.20	Default Panel 330x240	4	2.40	Default Wingnut	12	2019-04-04 11:00	2019-04-08 07:28	2019-04-06 11:00	70.0	2019-04-04 11:00	NULL
		Default Panel 330x120	2	1.20	Default Tie Rod	12						
		Wooden filling material	2	0.20	Default Coupler	16						
					Default PushPull Prop	6						
					Default Waler	0						
644734	6.20	Default Panel 330x240	4	2.40	Default Wingnut	12	2019-04-08 07:28	2019-04-09 11:57	2019-04-08 16:00	100.0	2019-04-08 11:00	2019-04-08 16:00
		Default Panel 330x120	2	1.20	Default Tie Rod	12						
		Wooden filling material	2	0.20	Default Coupler	16						
					Default PushPull Prop	6						
					Default Waler	0						

Figure 16: Linked Data Query Output Table for TCI Utilization Plan, Example

Then, for each data graph, an individual SPARQL query must select the required data and map it into a JSON format in order to work with it in Javascript. Afterwards, the algorithm has to calculate the required quantities of formwork for each wall instance and push the selected TCI elements to the respective wall object. The following figure reveals the output JSON object for a wall example with the calculated TCIs. The result of the calculation contains enough information to create a formwork utilization plan for a construction project.


```
{"wallinst": "http://test/walls/450d31df-4383-4692-9be4-9c0935e083ef-0008f0ba",
   "length": 6.2,
   "TCIsIn": [
          {"TCIinst": "http://test/tci/DefaultPanel330x240_0", "length": 2.4},
          {"TCIinst": "http://test/tci/DefaultPanel330x240_1", "length": 2.4},
          {"TCIinst": "http://test/tci/DefaultPanel330x120_0","length": 1.2}],
   "TCIsOut": [
          {"TCIinst": "http://test/tci/DefaultPanel330x240_0", "length": 2.4},
          {"TCIinst": "http://test/tci/DefaultPanel330x240_1", "length": 2.4},
          {"TCIinst": "http://test/tci/DefaultPanel330x120_0","length": 1.2}],
   "secTCIs": [
          {"TCIinst": "http://test/tci/DefaultCoupler", "weight": 4.58},
          {"TCIinst": "http://test/tci/DefaultTieRod", "weight": 4.43},
          {"TCIinst": "http://test/tci/DefaultWingnut", "weight": 2.58},
          {"TCIinst": "http://test/tci/DefaultPushPullProp", "weight": 22.8}],

   "TCIsCount": [
          {"TCIinst": "http://test/tci/DefaultPanel330x240", "Count": 4},
          {"TCIinst": "http://test/tci/DefaultPanel330x120", "Count": 2},
          {"TCIinst": "http://test/tci/TimberFilling", "Count": 2},
          {"TCIinst": "http://test/tci/DefaultCoupler", "Count": 4},
          {"TCIinst": "http://test/tci/DefaultTieRod", "Count": 8},
          {"TCIinst": "http://test/tci/DefaultWingnut", "Count": 8},
          {"TCIinst": "http://test/tci/DefaultPushPullProp", "Count": 8}],
   "TimberFilling": [
          {"TCIinst": "http://test/tci/TimberFilling", "Length": 0.2},
          {"TCIinst": "http://test/tci/TimberFilling", "Length": 0.2}]},
```


Figure 17: Output JSON Object of the developed program

In the output data, the object TCIsIn and TCIsOut represent the formwork instances for each side of the wall which are then enumerated with a running integer to be able to later address the individual form. This helps, for example, when including IoT-tracking of forms as one could trace each form back to which location and construction activity it belongs. In this case, the TCIs inside and outside of the wall are the same, as the wall is regarded as an individual object. The results of the program were then compared to the results from PERI Quicksolve for a similar wall design and thus, validated with a professional software solution. Both the professional calculation from PERI and the proposed prototype solution conclude in a realistic formwork layout for the given wall structure. However, the developed program in this stage of prototyping shows the following limitations compared to the professional formwork calculation.

- Corner elements need to be considered in the calculation by detecting if a wall has a corner ending. The wall-length has to be reduced respectively with the length of the corner elements if a corner is detected.
- If a wall has no corner, the ending must be closed with walers and plywood at the end of a wall sequence to prevent the concrete from pouring out. This has not been considered in the demo project.
- The plywood has the height of the formwork, the width of the concrete thickness, and an approximate thickness of 2 cm.
- Three walers shall be placed around the open wall ending to support the plywood and hold the concrete in place.

The listed shortcomings will be included in the further developing process of the program for the followed case study.

The final part of the program must write the data back into the triple store in order to create a combined graph which contains the data of all datasets as well as the calculated output data of the program. As described in the previous chapter, the data graphs are stored in individual named graphs within the dataset http://localhost:3030/TCI-Demo/ where the default graph is empty. The program is able to write the data to the default graph by creating an INSERT query that writes to the URL http://localhost:3030/TCI-Demo/update. As soon as the data is transmitted to the default graph, a new graph (http://localhost:3030/TCI-Demo/data/Combined) can be created by querying each named graph and adding them to the newly created combined data graph. The final dataset layout in Fuseki is shown below:

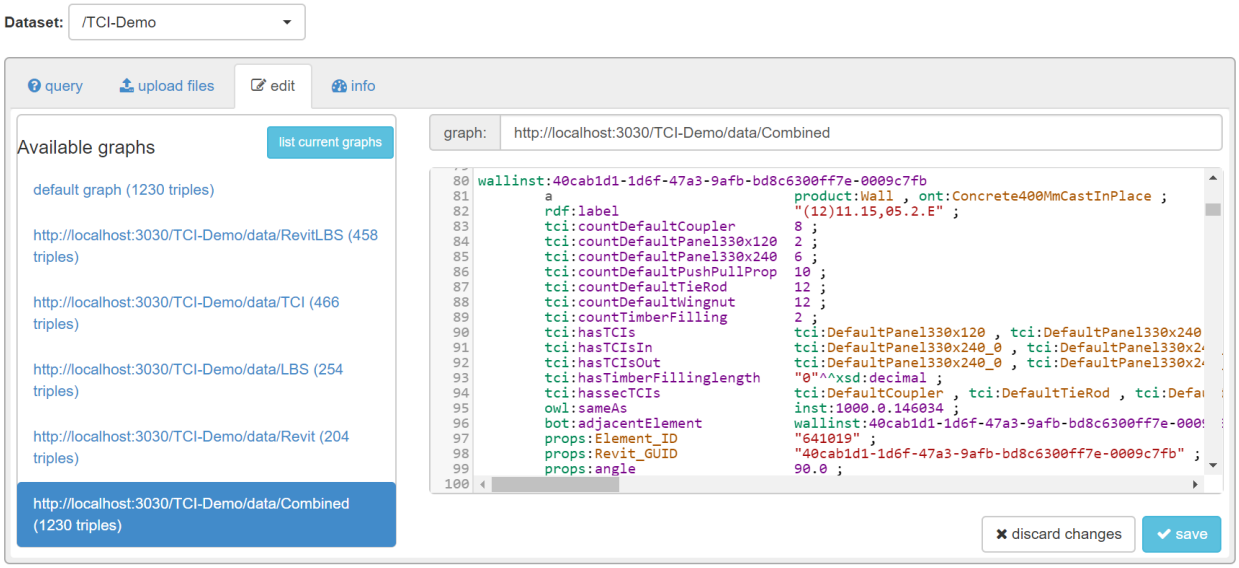


Figure 18: Final TCI-Demo dataset layout with named graphs

The created named graph is now containing a total of 1230 triples and has all the information to create the desired output table from figure 16. The combined data graph contains the individual datasets Revit, LBS, and TCI. It combines the building model elements with information about the schedule, the current progress as well as detailed information as quantities of TCIs that support the construction activities. By querying the data graph with a SPARQL-query, specific data can be obtained in a table format, ready for a data export in a convenient format. Using the following SPARQL-query against the combined data graph enables to narrow down the huge amount of data to the essential and desired outcome.

```
SELECT ?Element_ID ?length ?PrimaryTCIs ?TCIsCount ?SecondaryTCIs ?2TCIsCount
?Location ?PlannedStartDate ?PlannedEndDate ?ActualStartDate ?ActualEndDate ?P
rogressDate ?ProgressCompletion

WHERE {GRAPH <http://localhost:3030/TCI-Demo/data/Combined>
{
?Revitinst a product:Wall ;
    props:Element_ID ?Element_ID ;
    props:length ?length ;
    tci:hasTCIs ?PrimaryTCIs;
    tci:hassecTCIs ?SecondaryTCIs .

?1Countprop tci:iscounting ?PrimaryTCIs .
?Revitinst ?1Countprop ?TCIsCount .
?2Countprop tci:iscounting ?SecondaryTCIs .
?Revitinst ?2Countprop ?2TCIsCount .

  ?VICOinst a lbs:CompLoid ;
    props:Element_ID ?Element_ID ;
    lbs:hasLocation ?Location;
    lbs:taskPlannedStartDate ?PlannedStartDate;
    lbs:taskPlannedEndDate ?PlannedEndDate;
    lbs:taskActualStartDate ?ActualStartDate;
    lbs:taskActualEndDate ?ActualEndDate;
    lbs:taskProgressDate ?ProgressDate ;
    lbs:taskProgressCompletion ?ProgressCompletion .

FILTER (?VICOinst != ?Revitinst)
}}
```

Figure 19: SPARQL-Query against combined graph

The shown query was developed to create a narrowed down dataset in order to derive a TCI utilization plan. It shall provide a similar output as the desired output table in figure 16. First, the query is specifying the main subject, which in this case are the wall instances, from both the Revit and the VICO source. As the program **indexSameAs.js** was mapping the wall instances from both sources with the owl:sameAs relationship by the common Element ID, all information from both data sources can be queried and linked to each wall instance. The result of the simple query in figure 19 is then displayed in table format (CSV-file) and can be further utilized e.g. for data visualization. This output table is representing the main result of the demo project and serves as a proof of concept of the proposed solution as it contains each wall instance and information about when and where which types and number of TCIs are needed on site. An excerpt from the output table is can be found in table 4.

Revit		TCI					VICO		Exicute			
Element_ID	length	PrimaryTCIs	TCIs Count	SecondaryTCIs	2TCIs Count	Location	PlannedStartDate	PlannedEndDate	Actual StartDate	Actual EndDate	Progress Date	%
"645092"	"6.2"	"tci:DefaultPanel330x120" "tci:DefaultPanel330x240" "tci:TimberFilling"	"2" "4" "0.2"	"tci:DefaultCoupler" "tci:DefaultTieRod" "tci:DefaultWingnut" "tci:DefaultPushPullProp"	"4" "8" "8" "8"	"Lev1_loca(w)"	"2019-04-04 11:00:00.000"	"2019-04-08 07:28:48.000"	"NULL"	"NULL"	"NULL"	"0.0"
"585914"	"6.2"	"tci:DefaultPanel330x120" "tci:DefaultPanel330x240" "tci:TimberFilling"	"2" "4" "0.2"	"tci:DefaultCoupler" "tci:DefaultTieRod" "tci:DefaultWingnut" "tci:DefaultPushPullProp"	"4" "8" "8" "8"	"Lev1_loca(w)"	"2019-04-04 11:00:00.000"	"2019-04-08 07:28:48.000"	"NULL"	"NULL"	"NULL"	"0.0"
"640260"	"8.4"	"tci:DefaultPanel330x120" "tci:DefaultPanel330x240" "tci:TimberFilling"	"2" "6" "0"	"tci:DefaultCoupler" "tci:DefaultTieRod" "tci:DefaultWingnut" "tci:DefaultPushPullProp"	"8" "12" "12" "10"	"Lev1_loca(w)"	"2019-04-04 11:00:00.000"	"2019-04-08 07:28:48.000"	"NULL"	"NULL"	"NULL"	"0.0"

Table 4: Output table from querying combined graph

Chapter 5. Data Visualization and Distribution

The previous steps of prototyping a solution all intend to link existing data to a combined dataset that is able to provide a TCI utilization plan. After comparing the result with a professional calculation tool for formwork (PERI Quicksolve), it is assumed that the raw data is correct. Thus, the next crucial step of prototyping is to distribute the data to the target groups in the most convenient and accessible way. In this case, data visualization helps to bring the right amount and type of data to the specific target group.

As the output data is stored in RDF-triples in a triple store, there are several options on how to visualize the data, such as the use of a web application that directly queries the triple store. For the application in this thesis project, however, already existing and proven tools from the company Exigo A/S are used for data visualization and distribution. The first tool is a Power BI dashboard that is directly linked to a SQL database and receives updated schedule information through Exicute, the cloud platform with progress monitoring of Exigo A/S. As a second visualization tool, a new tab in the Exicute application is intended to display all required TCIs for each specific construction task. Similar to the "Quantities" tab, which shows the quantities of PCIs for each task, the new tab shall give updated information about the TCI utilization on site, broken down to the task level. The utilization of these visualization tools allows to integrate TCI information within an already established process, validated by the Danish construction industry. Thus, the adoption of the proposed solution within the construction industry is facilitated as it can come as an extended feature of an existing product.

5.1 SQL Conversion

In order to use both visualization tools, the data has to be converted into SQL format. For this purpose, a program (**indexRDF2SQL.js**) has been developed that receives the output data from the Fuseki triple store and writes it into common SQL tables via SQL Query language. Communication with the SQL database is enabled by the Microsoft SQL Server client for Node.js "mssql" which can be found with the following URL: https://www.npmjs.com/package/mssql#microsoft--contributors-node-v8-driver-for-nodejs-for-sql-server.

Based on the prototyping status of the solution, the following SQL tables are developed. These tables serve as the data source for the use of Power BI as well as the application Exicute, and thus must contain all necessary information to visualize the TCI utilization plan and meet the needs of the specific target group.

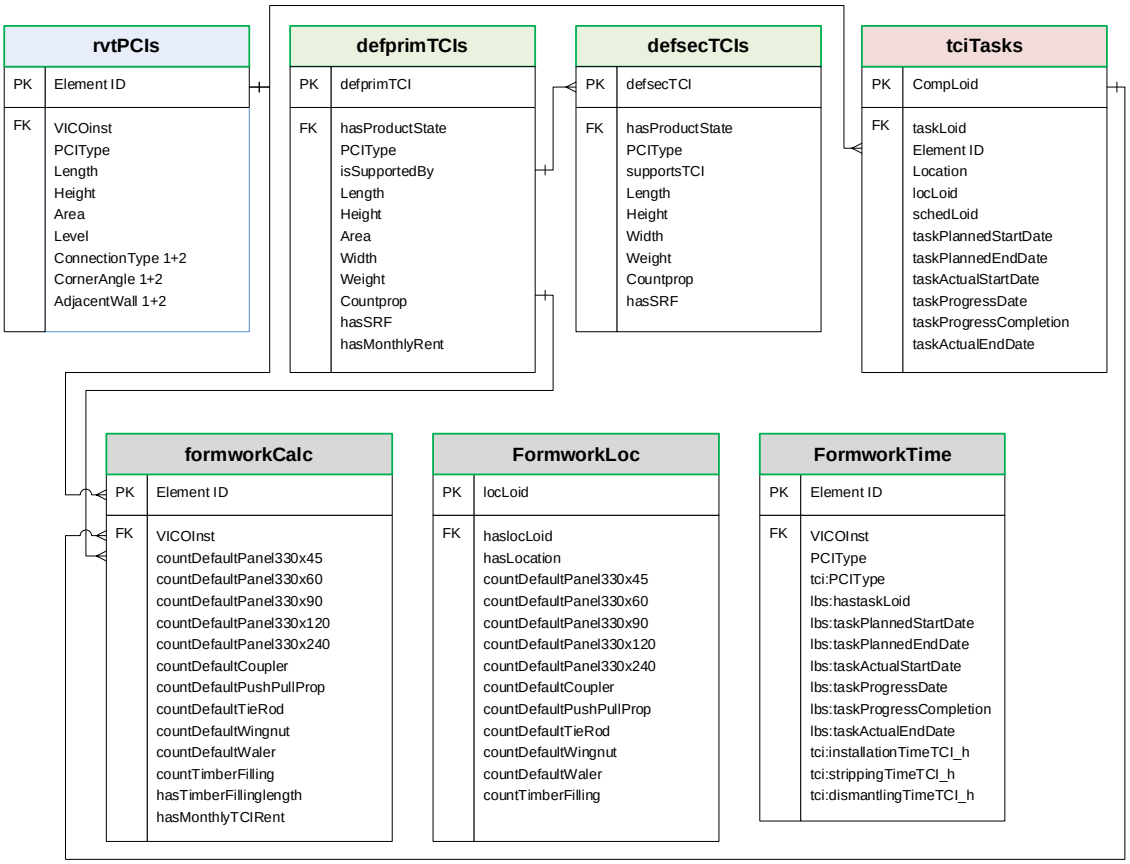


Figure 20: SQL tables for data visualization

5.2 Power BI

Power BI is a visualization tool from Microsoft that enables to build live dashboards, based on a huge variety of data sources, one being a SQL database. The direct link between the data and the dashboard allows to always have the updated data regarding the reviewed project. Power BI dashboards are mostly used to give a general overview of the current status and progress of the construction site regarding various aspects. Due to this superficial consideration, these dashboards are targeting the management perspective of a company.

Exigo A/S uses the dashboard among others to visualize the construction progress, a change order log, a risk calculation, a cashflow diagram as well as the resource allocation for the construction site. The following figure shows an example dashboard that is used as a template to create dashboards for visualizing the TCI data.

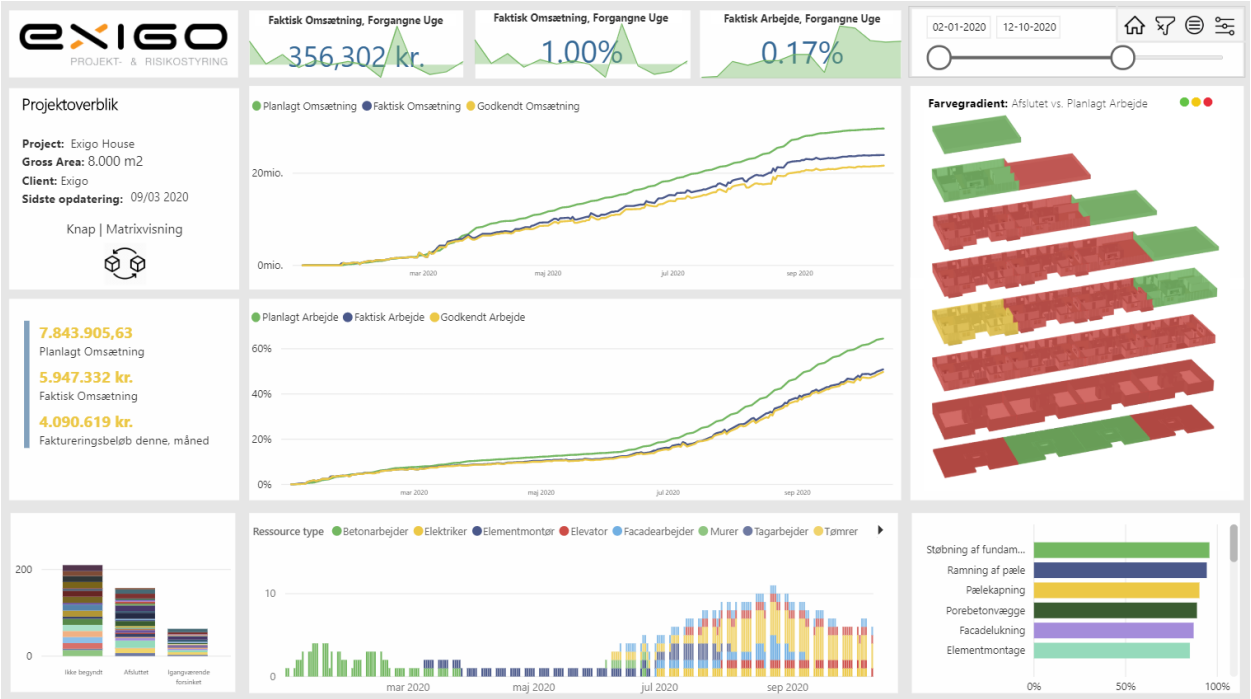


Figure 21: Example dashboard from Exigo A/S

In order to develop a new dashboard for the visualization of TCI data, some important aspects have to be considered. The following list provides all the features, the developed dashboard incorporates to visualize the TCI utilization plan.

- Exploded building view of the building model with different locations allowing to select specific locations
- Time slicer to specify the regarded period or specific date
- Selection tool to specify the specific TCI type to be reviewed in the dashboard
- TCI allocation graph showing the quantities of the TCI utilization on a time axis
- Comparison between static stock and dynamic stock (Static stock is calculated with the peak amount of TCI demand in the project and dynamic stock is representing the actual TCI demand with a buffer of 10%)
- Graph showing an accumulated cost comparison of TCIs on the construction site, based on the comparison between the static stock and dynamic stock.

The comparison between the dynamic and static stock is an exemplary quantification of the benefits, the proposed solution generates for the construction project. Here, the current practice is compared to the approach, a contractor could pursue with the proposed solution. In current practice, as also identified in chapter 5.1, formwork elements are ordered, based on an estimation of what would be the peak amount of elements the construction site needs to have in order to meet the demand from the construction tasks. This peak amount of elements is then delivered by the TCI provider and stored on site. As it is dimensioned to cover the peak amount of formwork elements, the utilization of the elements for most working days is relatively low, resulting in a waste of money and storage space because the big and unused formwork panels are lying around on the construction site. In case the formwork is rented, this static stock approach leads to high costs in rent, which would be avoidable with a more dynamic stocking approach. In order to enable a dynamic stock, transparency about the TCI utilization on the construction site is needed early in construction to order the elements from the supplier in advance and to allow a just-in-time delivery. As the proposed solution provides this transparency, the construction site would benefit from a more dynamic formwork stock, which is continuously adjusted to the changes in the demand. If there is a high demand for a short period, more items are delivered to the site in advance to cover the demand and as soon as the demand decreases again, the surplus of formwork elements is returned to the supplier. This results in a generally much lower stock on site, meaning waste of money and storage space. Thus, the rent saving potential of the proposed solution shows a clear and quantifiable benefit for the construction site.

In the developed dashboard, besides the TCI Allocation and Quantities, this is the central information that is derived from the utilization plan and visualized in the dashboard. In the dashboard, this information is divided into a utilization comparison between the dynamic stock and the static stock as well as a cost comparison, showing the cost development over time for both the static and dynamic stock. The resulting dashboard, incorporating all mentioned features is shown below.

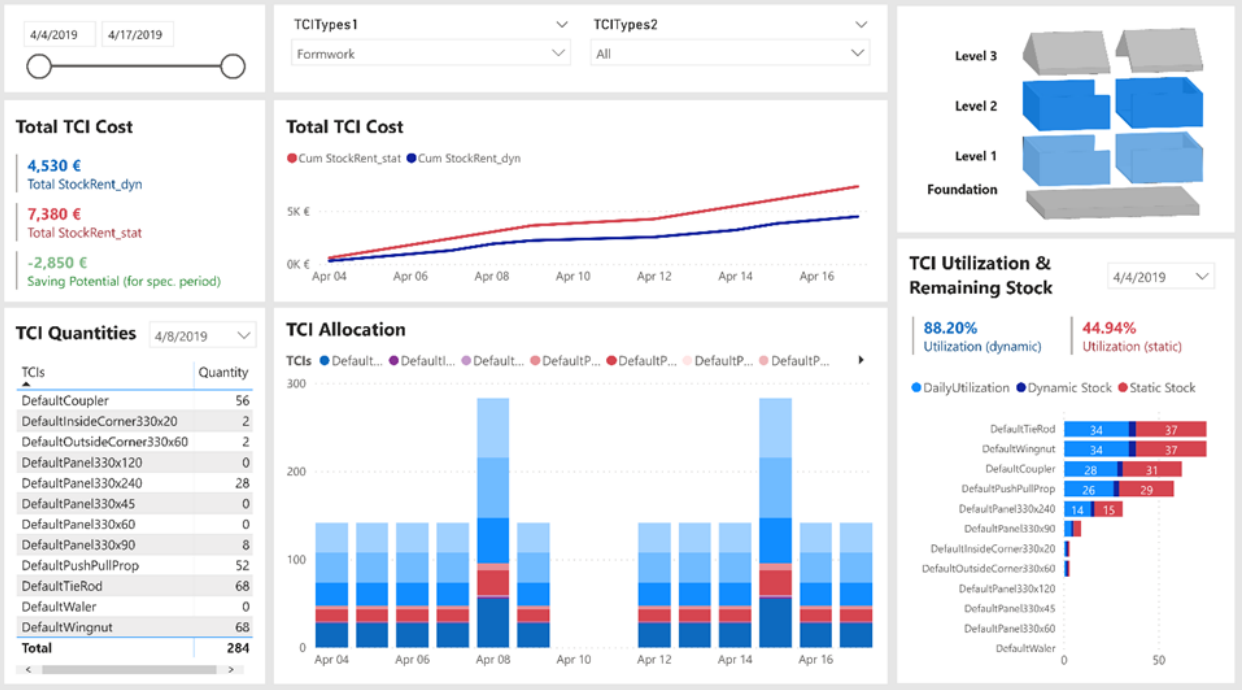


Figure 22: Demo Dashboard - *TCI Utilization*

5.3 Exicute Application

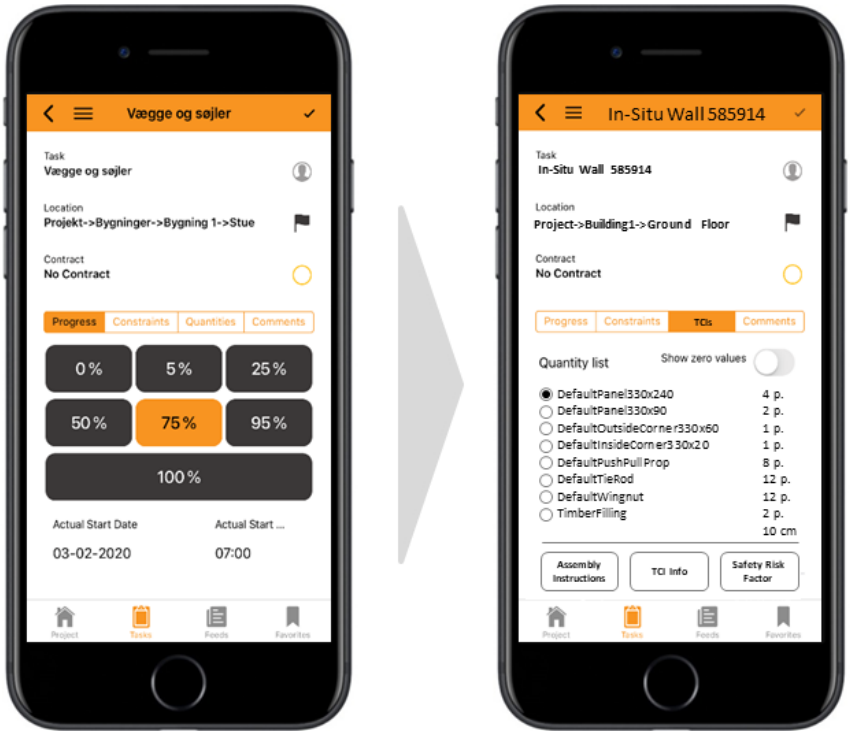


Figure 23: Interface of the Exicute App

The mobile application of Exicute is currently used to document the construction progress for each task. On a task-based level of detail, it shows the progress, constraints, quantities, and comments for each construction activity. A further development of the application with the implementation of the proposed solution could extend the shown information with a new tab, called “TCI Quantities”. This tab could then return the required TCIs for each individual task of the construction project. In this tab, the required TCI quantities can be shown along with more detailed information about each TCI, e,g, with formwork: its weight, the time it needs to be installed and dismantled, the location to store the items before usage, the location where to bring the items for the next task as well as a safety risk factor which reveals TCIs that require special attention when installed and utilized. Assembly instructions for each task could also be included from the supplier to ensure safe and correct utilization of the elements. Integrating the TCI utilization into Exicute would add an additional feature to the product which allows the user to better plan and manage TCIs on the construction site. As an additional feature, it could be sold to contractors or clients to increase productivity and safety through a transparent TCI consideration in the project.

Chapter 6. Bibliography

Costa, G., & Pauwels, P. (2015). Building product suggestions for a BIM model based on rule sets and a semantic reasoning engine. *32rd International CIB W78 Conference, Proceedings*, 98–107.

Martin, B., & Hanington, B. M. (2012). *Universal methods of design: 100 ways to research complex problems, develop innovative ideas, and design effective solutions*. Beverly, Mass.: Rockport Publishers.