Impact Lab - Zany Ants

A short report of the visualization project

Introduction

Xoom is a configuration management solution that helps users understand and manage the configuration of complex software systems, including dependency analysis, reporting, transport, revision control and comparison.

The configuration is composed of configuration items, each of which is a functional unit identified by one or more unique identities (aliases). Configuration items can be simple parameters or have a complex structure, sometimes containing other configuration items or even whole sets of configuration items of the same kind (types). They can also reference other configuration items from within various parts of their body, which creates a graph of dependencies.

This directed graph, which is neither necessarily acyclical (cycles will be common but rare) nor connected, can be visualised and browsed by the user of the software to explore the structure of the configuration and better understand it. Unfortunately, due to the number of relationships, the visualisation of all items, even when focused on a single item and organised organically by proximity, can quickly become overwhelming in its complexity and hence obscure rather than illuminate any insights that users might want to glean from it. Therefore, it is crucial to present the right amount of information with just the right amount of detail that is relevant in the current context.

To support Xoom, the project will aim to explore means of implementing interactive graph visualisation that will allow users to meaningfully and usefully interact with the graph. Both the visualisation from the point of view of the whole graph or a single item will be explored. The goal will be to allow users to decrease and increase the amount of detail shown based on their need by grouping items together using contextual grouping mechanisms that preserve the maximum amount of useful information while obscuring unnecessary detail. Also, the project may allow us to perform graph analysis or other machine learning techniques on the configuration.

Objective

In the project, we aim to visualize the configuration items in a more meaningful visualization. Hence, we are experimenting with some possibilities with grouping/clustering similar configuration items and minimize relations between them. Some tasks in this project are as followed:

1. Construct a graph from the initial dataset.
2. Graph analysis (e.g. degree, component, distance, etc.) of the dataset.
3. Visualization technique to present the data as a graph or other means of visualization that meets the project goals, both as a whole and from the point of view of a single item (i.e. single item's neighbourhood).

4. Interactive and informative visualization of configuration items and their groupings, and the relationships between them.

Delivery

By the end of the project, a set of deliveries will be produced, such as:

1. A Jupyter notebook that reads Xoom data, manipulates it and generates data for visualization
2. A set of files to generate interactive visualizations in the browser.
3. A short report about the project and all results during an investigation of visualizing information

Dataset

This project requires a dataset as a starting point to produce/generate all necessary datasets for visualisation purpose. The required dataset is the file **GetResultIndex.json**, which contains the details about the configuration items and their relationship.

Input

The input of this project is a JSON file named **GetResultIndex.json**. As a JSON file, it contains a key-value pair of each configuration item. This file has 2 main roots, “items” and “types”. The “items” root has a value of a list of configuration items, while the “types” root has a list of categories under which the configuration items belong to. However, the type of item can also be derived from the value of the itemId by extracting the characters *before* the character ‘[‘. For example in Figure 1, the item *‘ABCapacityLimits Index[District,TaskType]’* has a type *‘ABCapacityLimits Index’*. Hence, we focus more on the “items” root in constructing our item’s graph.

Figure 1. A snippet of the input JSON file, **GetResultIndex.json**.

An item can also depend on other items. This relation is captured in the key “dependencies”. Therefore in Figure 1, we can model the relation between the item *‘ABCapacityLimits Index[District,TaskType]’*, *‘ABCapacityLimits Property[District]’*, and *‘ABCapacityLimits Property[TaskType]’* with the arrows from *‘ABCapacityLimits Index[District,TaskType]’* to *‘ABCapacityLimits Property[District]’* and *‘ABCapacityLimits Property[TaskType]’* as shown in Figure 2.

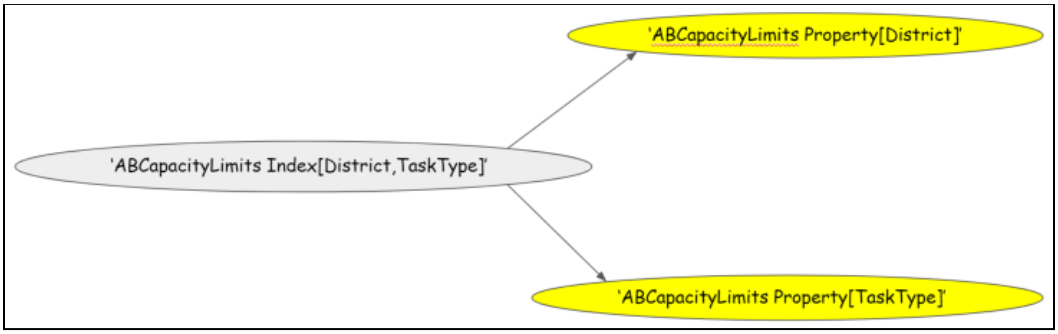

Figure 2. An example of representing a configuration item into a directed graph.

Finally, an item can also have multiple ids. For instance, the item *'Setting[[Application]|Decomposition|Bundler BGO Region 1|]'* that is depicted in Figure 3, is also known as *'Decomposition[Bundler BGO Region 1]'* and *'StandardSetting[[Application]|Decomposition|Bundler BGO Region 1|]'*. In this case, the relationship will be modelled in Figure 4.

```
{
   'itemId': 'Setting[[Application]|Decomposition|Bundler BGO Region 1|]',
   'identities': [
      {'fields': {'Category': 'Decomposition', 'Name': '', 'Owner': '[Application]',  'SubCategory': 'Bundler BGO Region 1'},
       'id': 'Setting[[Application]|Decomposition|Bundler BGO Region 1|]', 'type': 'Setting'},
      {'fields': {'SubCategory': 'Bundler BGO Region 1'},
       'id': 'Decomposition[Bundler BGO Region 1]', 'type': 'Decomposition'},
      {'fields': {'Category': 'Decomposition', 'Name': '', 'Owner': '[Application]', 'SubCategory': 'Bundler BGO Region 1'},
       'id': 'StandardSetting[[Application]|Decomposition|Bundler BGO Region 1|]', 'type': 'StandardSetting'}
   'dependencies': ['Engineer Index[Region]',  'Region[1 - Regina]',  'Task Index[Region,DueDate]'],
   ],
}
```

Figure 3. An example of a configuration item with multiple ids.

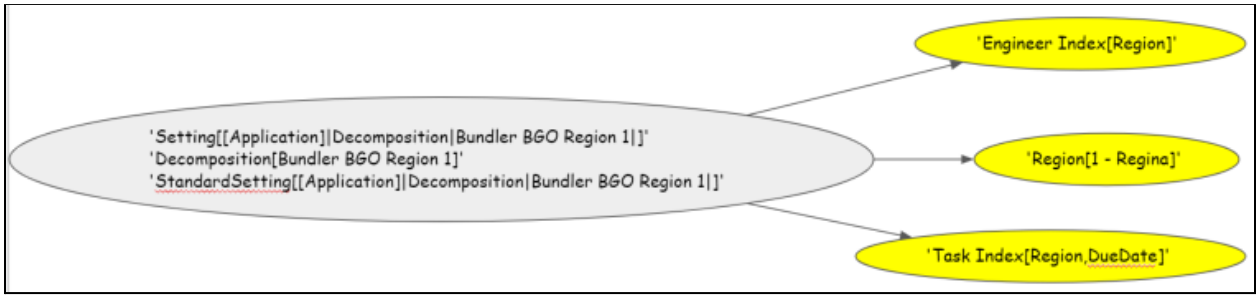

Figure 4. An example of representing a configuration item with multiple ids in a directed graph.

Output

At the end of the project, some deliverables that are produced. Those files have been put online as well, for Zany Ants to download and to demonstrate the visualisation. Those deliverables are as followed:

1. An interactive graph representation of the configuration items (.html, .json, .js, .css).
2. A source code of how a graph is constructed and visualized (.ipython).
3. A short report describing the project and the visualization process during the project (.pdf).

Process

In order to generate all output files of visualization, we first generate a directed graph between all items (we name this graph **G_Item_nx** in our code). This graph represents the whole connectivities between items in a system. From this graph, the in and out components of each node are generated. On one hand, the in-component (of node A) is the set of all nodes from which there is a directed path to node A, including node A itself. On the other hand, the out-component (of node A) is the set of nodes that are reachable via directed paths starting at a specified node A, and including A itself. Therefore, each individual item has 2 corresponding JSON files to be visualized as its in and out components. In addition, for these 2 JSON files, there are additional 2 JSON files in a different location that represent the simplification of the in and out components of each item, by removing some edges and nodes.

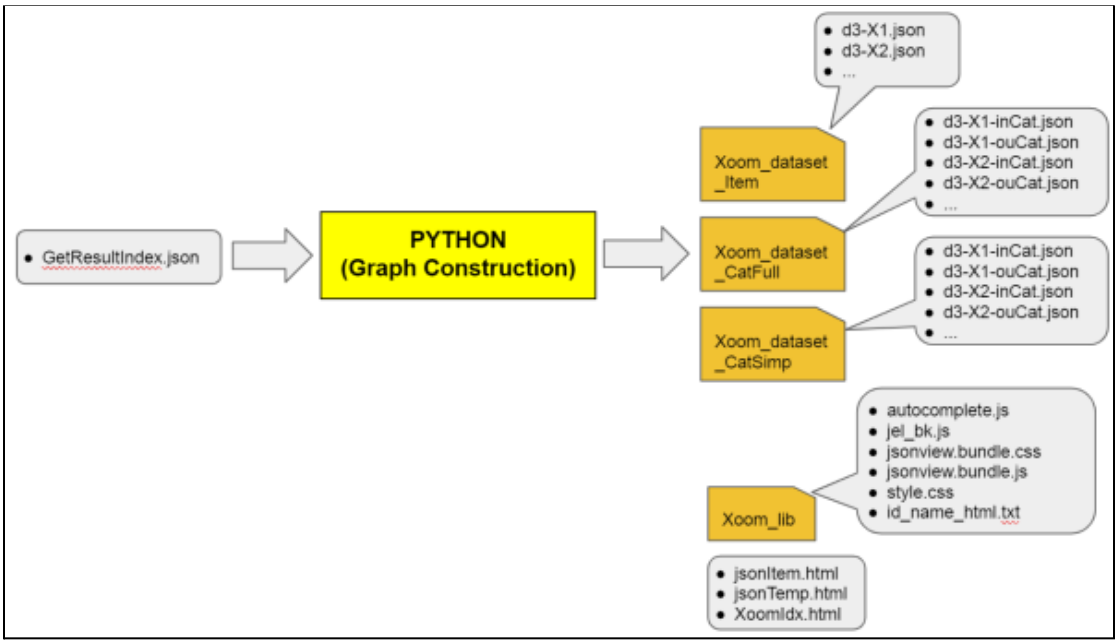

Figure 5. A skeleton of the project's files structure.

In general, the main file is the **XoomTools_Final.ipynb**. Through this file, all the output files are generated based on the JSON input file, **GetResultIndex.json**. The first step that we do is to map the XoomId (it is called '*idX*' or *'POI'* in the **XoomTools_Final.ipynb**) with the "itemId" based on the "items" root in the **GetResultIndex.json**. This mapping schema of the XoomId is stored as a dictionary called

'dictAKA' in the **XoomTools_Final.ipynb**. Figure 6 shows the mapping between XoomId and itemId. Note that for an item with multiple ids, it has the same XoomId.

```
'X182': ['Agent Type[Sync RosterShift with Calendar Agent]'],
'X183': ['Agent Type[Workflow Agent]'],
'X184': ['AgentOperation Index[AgentName]'],
'X185': ['AgentOperation Index[AgentName,Status]'],
```

Figure 6. A snippet of the mapping schema, '*dictAKA'* in the **XoomTools_Final.ipynb**.

Building a directed graph

Based on the mapping schema (in the *'dictAKA'*) and the input JSON file, **GetResultIndex.json** (for its information about the relations between items), we generated a directed graph between the configuration items. We name this graph **G_Item_nx** in our code. In this graph, a node represents a single item and an edge represents a dependency between items.

The naming of the nodes will start with "X". For example, "X183" represents the item *'Agent Type[Workflow Agent]'* (see Figure 6). In cases where an item has multiple ids, it will be represented as the same node. For example node "X10286" has 3 ids and all of them are captured in a node's attribute *'itemId'*. In other words, the attribute *'itemId'* in a node is the same as the *itemId* in the input JSON file. This is shown in Figure 7.

```
POI = "X10268"
G_Item_nx.nodes[POI]

{'itemId': ['Setting[[Application]|Decomposition|Bundler BGO Region 1|]',
  'Decomposition[Bundler BGO Region 1]',
  'StandardSetting[[Application]|Decomposition|Bundler BGO Region 1|]']}
```

Figure 7. Accessing a node in the directed graph, **G_Items_nx,** with multiple ids.

The dependency between items in the directed graph is modelled by taking the *dependencies* information in the input json file as shown in Figure 8 (left). The generated graph in Figure 8 (right) also shows the arrows from the node "X10286" to nodes "X5826", "X3221" and "X3457", where:

- "X10286" is the item with 3 ids:
  - *'Setting[[Application]|Decomposition|Bundler BGO Region 1|]'*,
  - *'Decomposition[Bundler BGO Region 1]'*,
  - *'StandardSetting[[Application]|Decomposition|Bundler BGO Region 1|]'*
- "X5826" is the item with id: *'Engineer Index[Region]'*
- "X3221" is the item with id: *'Region[1 - Regina]*
- "X3457" is the item with id: *'Task Index[Region,DueDate]'*

```
{'dependencies': ['Engineer Index[Region]',
  'Region[1 - Regina]',
  'Task Index[Region,DueDate]'],
 'identities': [{'fields': {'Category': 'Decomposition',
    'Name': '',
    'Owner': '[Application]',
    'SubCategory': 'Bundler BGO Region 1'},
   'id': 'Setting[[Application]|Decomposition|Bundler BGO Region 1|]',
   'type': 'Setting'},
  {'fields': {'SubCategory': 'Bundler BGO Region 1'},
   'id': 'Decomposition[Bundler BGO Region 1]',
   'type': 'Decomposition'},
  {'fields': {'Category': 'Decomposition',
    'Name': '',
    'Owner': '[Application]',
    'SubCategory': 'Bundler BGO Region 1'},
   'id': 'StandardSetting[[Application]|Decomposition|Bundler BGO Region 1|]',
   'type': 'StandardSetting'}],
 'itemId': 'Setting[[Application]|Decomposition|Bundler BGO Region 1|]'}
```


Figure 8. A configuration item in the input json file, **GetResultIndex.json**, and how it is represented in the directed graph, **G_Item_nx**.

With this principle, **G_Item_nx** has a total 10,591 nodes and 25,317 edges. In addition, it is a disconnected graph as some nodes that are not connected with the other nodes. Figure 9 shows some information about graph **G_Item_nx**, including the distribution of its islands (or component in the graph theory).

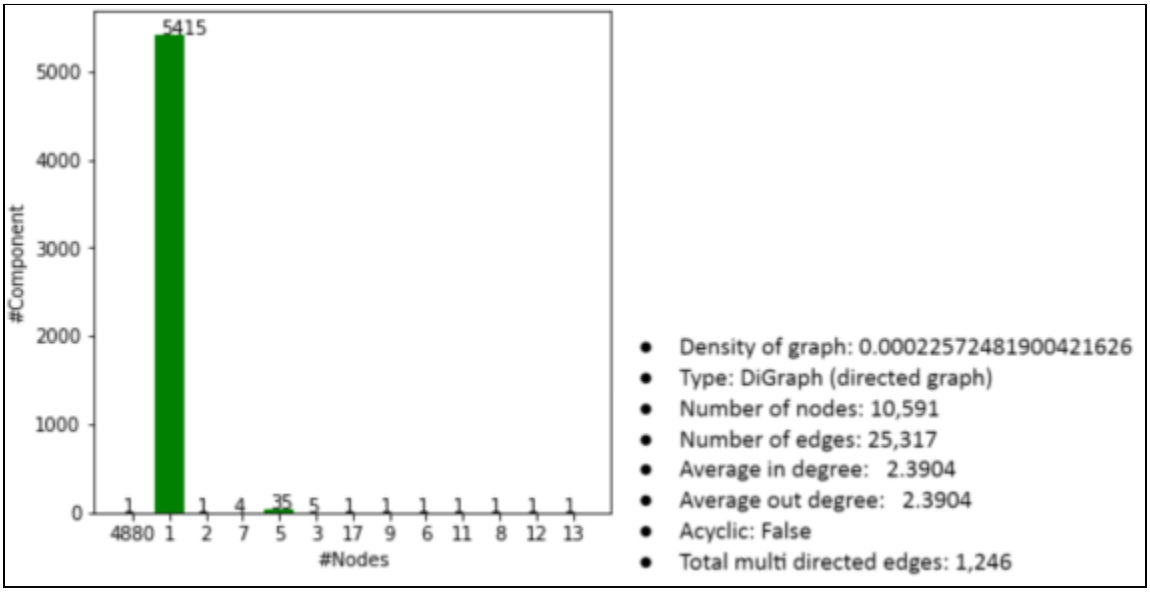

Figure 9. A characteristic of the generated directed graph, **G_Item_nx**.

Based on this directed graph, we construct the in and out component for each node or each configuration item..

Generate files in the Xoom_dataset_Item folder

A set of json files in this folder follows a similar structure with the input JSON, **GetResultIndex.json**. Here, the naming procedure is similar to the way we name the nodes in the graph **G_Item_nx**. For example, **d3-X20.json** contains information about the item *'ABCapacityLimitsDynamic Property[EntryDate]'* only taken from our input JSON file, **GetResultIndex.json.** This can be seen in Figure 10 (right), where it can be accessed by clicking the hyperlink of the node in Figure 10 (left). By representing each item in a single JSON file, will simplify the visualization of the individual item. To visualize each item, we use tree structure visualization as JSON format suits very well with this representation.

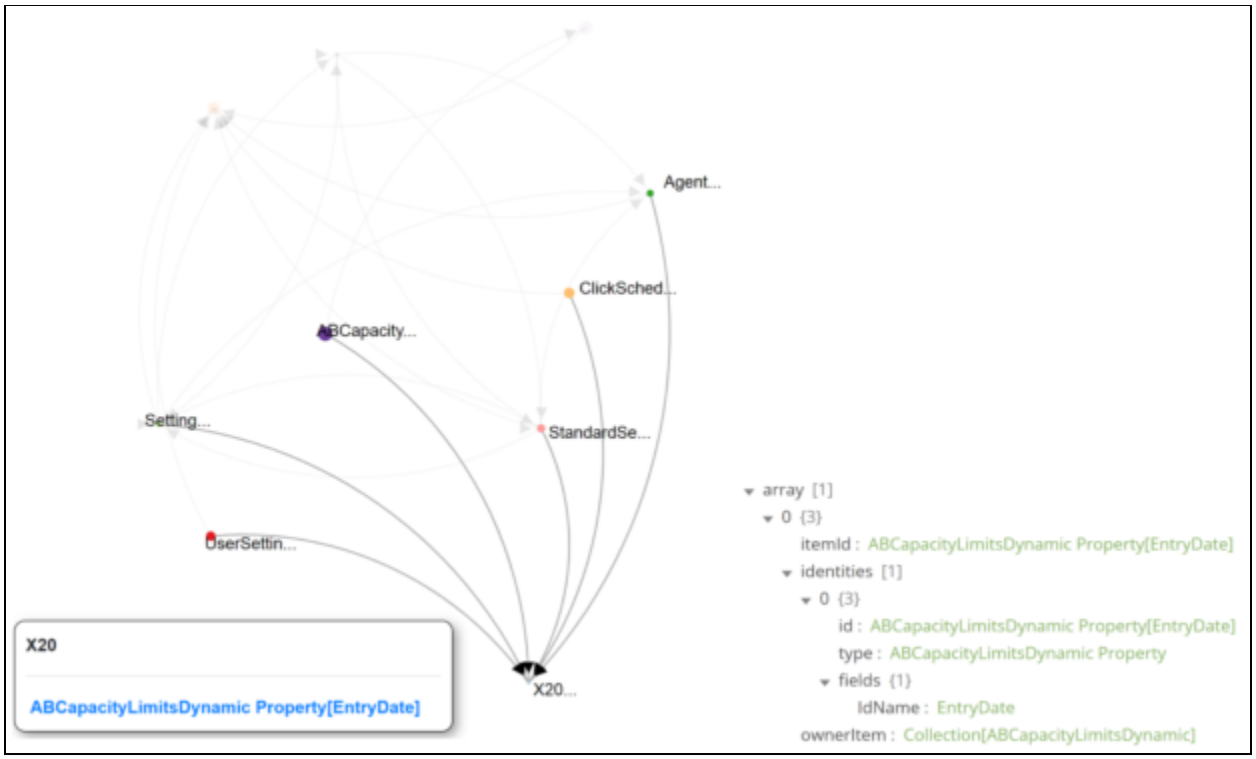

Figure 10. The visualization of the item "X20" in its in component graph (left) and its details in tree structure representation (right).

Generate files in the Xoom_dataset_CatFull folder

In this folder, we generate the full in and out components of all nodes in graph **G_Item_nx**. Each node will have 2 associated JSON files representing its in and out components. The naming of the files also indicates which node has which in or out components. For example, node "X6", which is the item *'ABCapacityLimits Property[Key]'*, has in and out components in the file **d3-X6-inCat.json** and file **d3-X6-ouCat.json** subsequently.

In constructing the in and out components, we decided to combine/summarize the nodes into their type (or category) in order to reduce their complexity. Hence, the nodes in the generated in and out components represent the types of items. Note that this modelling is different from the nodes in the graph **G_Item_nx**, where a node represents an item. Thus, all items with the same types are collapsed into the same node in the in and out components. Figure 11 (left) and Figure 11 (right) show the in and out components of node "X6" consecutively.

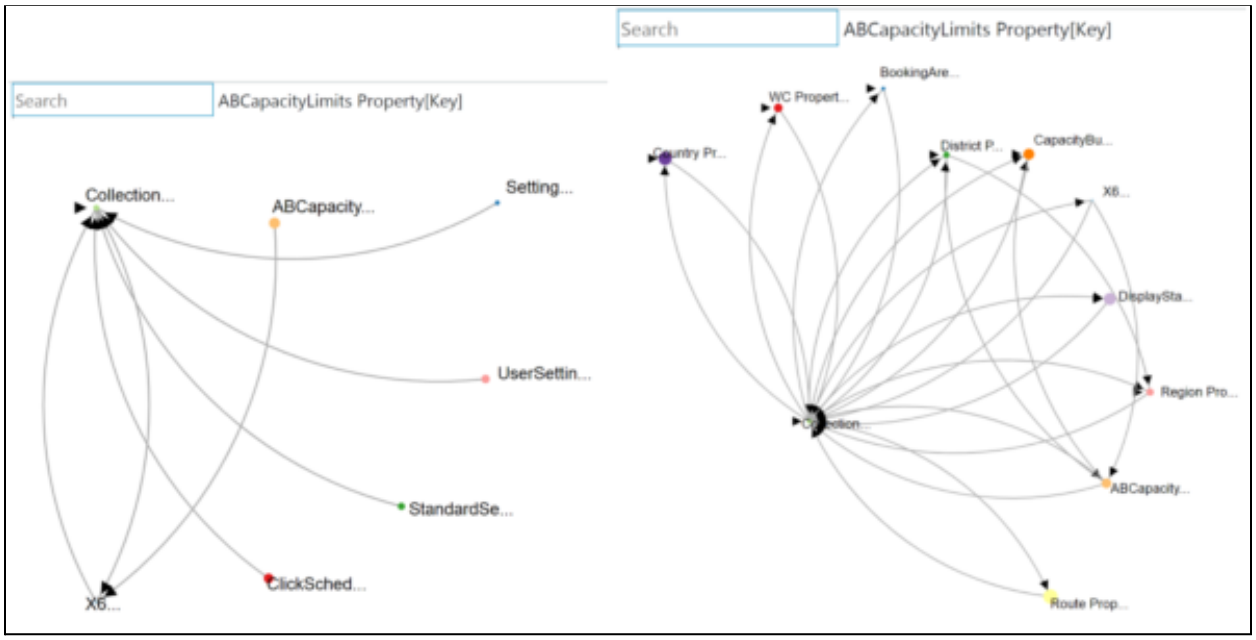

Figure 11. The visualization of the item "X6" in and out components in a full version.

Although we collapse the nodes with the same types, we make an exception for the node in focus. This node will stand out among the other nodes to show that we are focusing on the in and out components of it. From Figure 11, the focus node is "X6".

Generate files in the Xoom_dataset_CatSimp folder

Essentially, the JSON files in this folder are similar to the JSON files in the folder **Xoom_dataset_CatFull**. Also, the naming procedure is the same. The only difference is that the in and out components in this folder are (sometimes) less complex because we remove several edges. As a starting point, we remove all outgoing edges from the types "Collection", "Setting", and "Index". The outgoing edges from these types are removed because we are not interested in the nodes that they are pointing to. In other words, we are not interested in their neighbour (in a directed graph, a neighbour of node A is a node that node A is pointing to, and NOT a node that is pointing to node A). Figure 12 (left) and Figure 12 (right) show the difference between the in and out component of the focused node "X6" in its Simple visualization. Compare with Figure 11 (left and right), Figure 12 (left and right) are simpler.

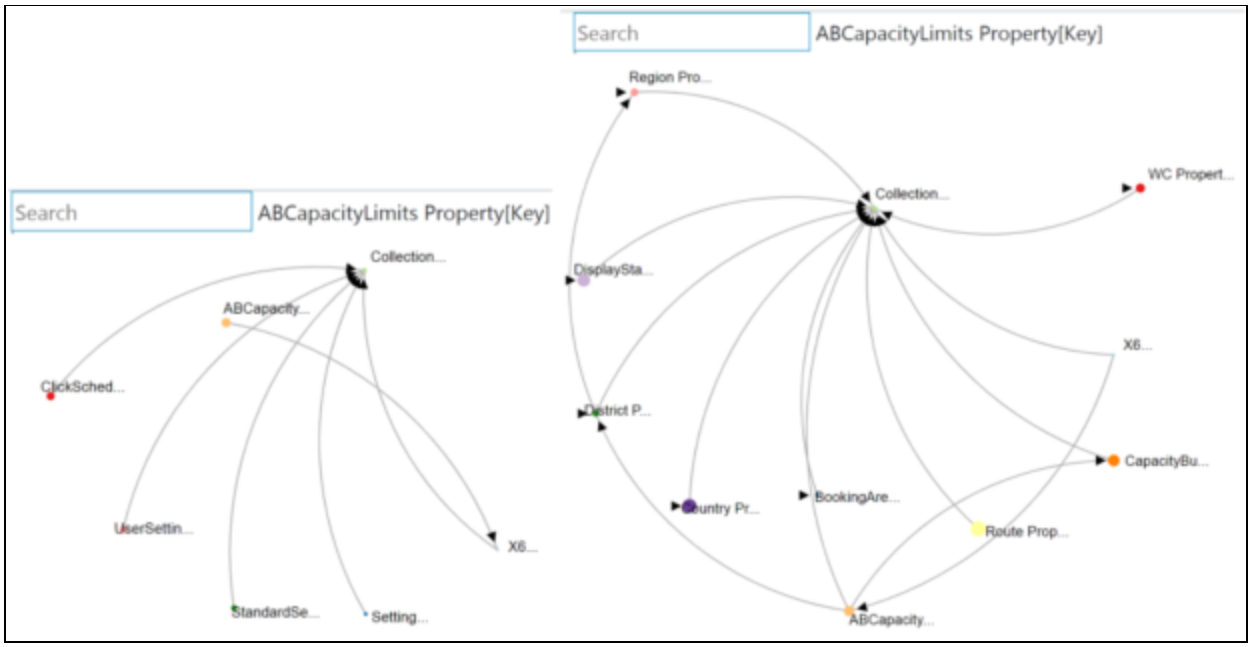

Figure 12. The visualization of the item “X6” in and out components in a simple version.

Generate the id_name_html file

Finally, the last file we generate is the **id_name_html.txt**. This file is used to show the hyperlinks of visualising the in and out components of the focus item. Each row in this file is designed with the HTML tags and represents an individual item with its in and out components (Full and Simple version). Ultimately, this file will be loaded by the file **XoomIdx.html** as an initial page to explore the visualization. Figure 13 shows the snippet of the file **id_name_html.txt**.

```
<li>
    ABCapacityLimits Index[District,TaskType] ||
    Full
        (<a href='jsonTemp.html?cat=ABCapacityLimits Index[District,TaskType]&JSON=Xoom_dataset_CatFull/d3-X1-inCat.json' target='_blank'>in 1</a>)
        (<a href='jsonTemp.html?cat=ABCapacityLimits Index[District,TaskType]&JSON=Xoom_dataset_CatFull/d3-X1-ouCat.json' target='_blank'>ou 11</a>) ||
    Simple
        (<a href='jsonTemp.html?cat=ABCapacityLimits Index[District,TaskType]&JSON=Xoom_dataset_CatSimp/d3-X1-inCat.json' target='_blank'>in 1</a>)
        (<a href='jsonTemp.html?cat=ABCapacityLimits Index[District,TaskType]&JSON=Xoom_dataset_CatSimp/d3-X1-ouCat.json' target='_blank'>ou 11</a>)
</li>
<li>
    ABCapacityLimits Property[District] ||
    Full
        (<a href='jsonTemp.html?cat=ABCapacityLimits Property[District]&JSON=Xoom_dataset_CatFull/d3-X5-inCat.json' target='_blank'>in 8</a>)
        (<a href='jsonTemp.html?cat=ABCapacityLimits Property[District]&JSON=Xoom_dataset_CatFull/d3-X5-ouCat.json' target='_blank'>ou 9</a>) ||
    Simple
        (<a href='jsonTemp.html?cat=ABCapacityLimits Property[District]&JSON=Xoom_dataset_CatSimp/d3-X5-inCat.json' target='_blank'>in 8</a>)
        (<a href='jsonTemp.html?cat=ABCapacityLimits Property[District]&JSON=Xoom_dataset_CatSimp/d3-X5-ouCat.json' target='_blank'>ou 9</a>)
</li>
```

Figure 13. A snippet of file **id_name_html.txt** to provide a set of hyperlinks in the file **XoomIdx.html**.

Visualization

When all files are properly generated, we can start exploring the visualization of the items and their connectivities in the system. Several steps for us to start exploring the

1. Point to the **XoomIdx.html** through a browser.

2. **XoomIdx.html** will show a set of hyperlinks from the **id_name_html.txt**. As mentioned, each row represents an item and shows its in component or out component. There are also links to visualize the Full and Simple version of the focus item.
    a. Full means that we visualize all incoming and outgoing edges of all nodes in the in and out components of the focus item.
    b. Simple means that we remove outgoing edges from the nodes with the type 'Collection', 'Property', and 'Index' in the in and out components of the focus item.
3. When the hyperlink is clicked, the in or out component of a focused item will be shown through the file **jsonTemp.html**. This file will load a corresponding JSON file in the folders **Xoom_dataset_CatFull** (for full version) and **Xoom_dataset_CatSimp** (for a simple version).
4. The **jsonTemp.html** interactively visualises the in or out component of the focus item. Again, a node in this visualisation represents a type (not an item). When we click on a node, it will highlight its neighbour (direct connection to the node) and blur the other dependencies (the indirect connection to the node).
5. In addition, each node contains the items with the same types when we click on it. Thus, it leads us to file **jsonItem.html** that shows the visualization of an individual item. This visualization is in a tree structure format from the JSON file in the folder **Xoom_dataset_Item**.

Conclusion and Future Works

This demonstrates the use of graph representation in visualizing relation between configuration items. As an experiment, we have tried several approaches in summarizing the information for a simple and interactive visualization. We have also generated a .gefx file to be loaded in the Gephi open-source application for graph visualization and analysis. Gephi has plenty of out-of-shelf techniques for exploring, manipulating, filtering graph, which is worth trying. This will open another opportunity to integrate Gephi for visualization of the configuration items.

At this time, we only have one type of relationship between items, which is *dependency*. Hence, the arrows in our graph essentially have the same meaning. Further, we may be able to develop richer ontologies to differentiate relations. We believe that this can contribute to the simpler visualization.

(Code) References

- https://networkx.org/
- https://gephi.org/
- https://github.com/d3/d3-force
- https://embed.plnkr.co/plunk/SawtfmR9iZvlLWue