Configuration Tool Design Notes

**Design Document**

Table of contents

[1 Introduction 3](#_Toc413049795)

[1.1 Purpose of this document 3](#_Toc413049796)

[1.2 Scope 3](#_Toc413049797)

[1.3 Constraints 3](#_Toc413049798)

[2 Design overview 4](#_Toc413049799)

[2.1 General considerations 4](#_Toc413049800)

[2.1.1 UI Style 4](#_Toc413049801)

[2.1.2 Initial Navigation Pane 4](#_Toc413049802)

[2.1.3 Folder Contents 4](#_Toc413049803)

[2.1.4 Further UI Guidelines 6](#_Toc413049804)

[2.2 Role Oriented Configuration 7](#_Toc413049805)

[2.2.1 Role UIC 7](#_Toc413049806)

[2.2.2 Data Mapping UIC 7](#_Toc413049807)

[2.2.3 Parameter Mapping UIC 8](#_Toc413049808)

[2.2.4 Node UIC 8](#_Toc413049809)

[3 Business Logic 8](#_Toc413049810)

[3.1 Elementary integrity rules 9](#_Toc413049811)

[3.1.1 Not NULL constraint 9](#_Toc413049812)

[3.1.2 CHECK constraint 9](#_Toc413049813)

[3.1.3 Unique Key constraint 9](#_Toc413049814)

[3.1.4 Foreign Key constraint 9](#_Toc413049815)

[3.2 Advanced integrity rules 9](#_Toc413049816)

[3.2.1 Non-cyclic component dependencies 9](#_Toc413049817)

[3.2.2 Proper component layering 10](#_Toc413049818)

[3.2.3 Unique parameter names 10](#_Toc413049819)

[4 Glossary 10](#_Toc413049820)

[5 References 10](#_Toc413049821)

[6 Change History 11](#_Toc413049822)

# Introduction

## Purpose of this document

This document contains several review items of the ui\_draft.pdf document and delivers some important design requirements to take into account.

## Scope

Configuration Tool initial version

## Constraints

n.a.

# Design overview

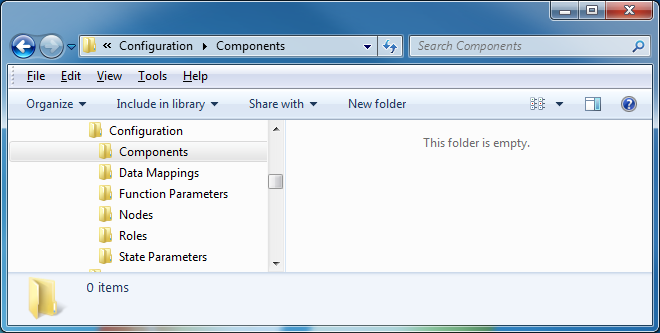
## General considerations

### UI Style

It is ok to implement the application by using the graphical style presented in the initial document.

### Initial Navigation Pane

Initial proposal silently assumes a specific workflow the user would take – start with nodes and work through to the items of interest based on a predefined window hierarchy. Our opinion is that the tool should not impose any workflow. The best initial screen should just contain the list of virtual folders corresponding to major database entities, just like this here:



Clicking onto a folder displays its contents.

I agree with NOT constraining the user, this also is reflected in the original design; the initial navigation is in the “View” drop-down. A folder concept should not be used in the planned app, because there are no folder semantics, especially you cannot copy or move items between folders. But the initial overview is a good point and possible. For this, there are two options: Make a specialized entry screen which lists all top level objects (possibly with counts or some context) or extent the View menu to include all top level objects. In this case, Roles could be the default selection. I would prefer this second option. The new UI draft shows an “Object” menu and a “View” menu. Object contains all top-level objects, as shown in the screen shot above. View currently contains “Table” and “Panels”. See the sheet “Roles” in “ui-draft-2” for details.

### Folder Contents

As a folder actually corresponds to the underlying database table, handling the folder contents depends on the kind of entity. A specific UI component should be implemented to optimally handle the actual entity contents and semantics. However all major UI components (UIC) should follow the same essential subcomponent architecture:

* Search form that can allow flexible search for entities in accordance with various parameters
* Master result list that displays entities matching the current search criteria and allows result list paged navigation (like in a google search)
* Detail form that displays the common attributes and meaningful relationships of the current entity selection

The current structure already follows this pattern, with the exception, that the search is only a single field at the top of all tables. I suggest to extent this so that a full search is available. It must be clarified if this should be opened from the main menu or from a button in each form.  
In the old design, the Panel list screens had the role of a master list. In the new design, table views also are possible, these are the master result lists. In the new design, the quick filter and the search button always are visible. A prototype has been implemented and can be shown.

#### Search Form

Since an SQL database provides unlimited search means also a search form for the given type of entity may be arbitrarily powerful. We have yet to decide individually about the layout and capabilities of each single search form.

Yes. Although some objects have very few fields of their own, so for such simple objects a single search field might be sufficient.

Other topic: I heavily recommend giving each top-level object its own name field. This is more consistent and much more user friendly.

#### Master Result List

Displaying entities as panels alone is too limited. Agreed! The new design also allows table views. Other vies might be possible, but the implementation effort would be increased. Just like in the Windows File Explorer, support for list/hierarchy style and details style display has to be provided. Tables clearly are possibly, simple Lists as well. Hierarchic display has to be discussed. The existing design already shows hierarchies for some objects.

In the details style representation sorting of contents by column caption click has to be supported. To support sort by multiple columns, click at the column A caption after a click at the column B caption works as if we sort by pair (A, B). This feature is very handy. The prototype indicates this feature by a little sort indicator to the right of the table column header.

Other layouts shall offer a multi-column sort facility. This usually must be accompanied by holding the Shift key. But presence of a keyboard cannot always be assumed.

List/hierarchy style is helpful to display logical hierarchies that is required for some types of entities (i.e. function parameters that often build hierarchies based on qualified names). It also may be helpful in the future to display entities that form natural hierarchies.

Would indentation of tabular entries be sufficient?

A master result list should support multiple selection of entities, (un-)select all as well as the standard copy/cut/delete operations applied to the current selection. The exact semantics of copy/cut/delete has yet to be defined individually on the per-entity basis. Most essentially, we have to define how to handle the relationships of the object(s) in question.

Yes, multiselect is already present in the design. It can easily be extended to tabular display. The new design allows this, the prototype supports view switching.

Copy/Paste is more critical because of object hierarchies. If possible, it can be represented by buttons in the toolbar.

#### Detail Form

Allows to edit contents of the (possible multiple) entity selection. Contents are not only attributes of the entity but also meaningful relationships which are yet to be defined depending on the type of the entity.

Current design aready has this.

For the display of the relationships **the master list of the corresponding UIC shall be reused**, here filled with the results of the corresponding search (here an AND filter by the given user selection). Default read-only display of a relationship view shows the current relationships and a button to switch to the edit mode where the current relationships are displayed as selected and where the user can extend or shrink the selection as well as apply the corresponding search form to comfortably find objects to add.

Already designed in a very similar way. Details must be discussed.

#### Screen Layout

It is not necessary to display all 3 subcomponents on the same screen. The search form may be hidden initially or better be displayed in the “simple” form (with a switch to the advanced mode). The detail form may be displayed on demand in a separate screen (but its origin screen then shall be displayed in the navigation history – s. b.).

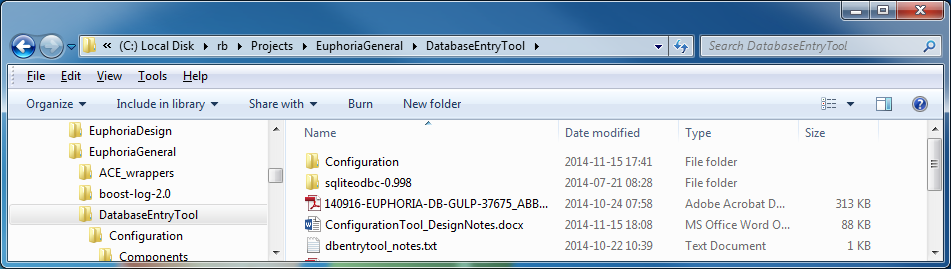
Agreed. Current design already has this. I would suggest using 2 default levels: Master and Detail with Advanced-Search shown on request.

History currently is planned only as a Back button. If desired, the whole history stack could be shown. This could be done as drop-down (like in most browsers) or as breadcrumbs (as in the screen shot and many web sites.)

As a detail form will display relationships of the given master list selection we shall allow seamless navigation in the database by selecting the relationship object (in the to-one case) or a subset of the relationship list (in the to-many case) and opening the corresponding component.

This is OK for me. The history stack might get quite large, though.

The complete current navigation history path shall be tracked in the address panel similar to this here:



This allows the user to easily jump back to an earlier step (at the left side) which also forgets the partial navigation sub-path to the right from the current location.

A real address bar is not recommendable, but either breadcrumbs or a drop down, as said before.

### Further UI Guidelines

* Enumeration types shall be displayed with their names and in detail forms by means of a choice control (combo box or similar).
  + OK
* Single-Level undo shall be supported for all editing operations by the user
  + We already planned a Cancel button. The effort for full undo/redo should not be much higher than for a single level. But the effort for Undo/Redo always is much higher than for a simple Cancel button.
* As many as possible errors caused by the latest user action and detected by the tool shall be reported inline (optimally highlighted) to allow fast correction actions.
  + We planned a message section at the top. But inline might be possible as well.
* Tooltips should be enabled for all screen elements and be editable without the need to reassemble the configuration tool.
  + Configurable in a separate file? The effort is higher but it is possible.
* No modal elements except emergency alerts and Save/New/Open boxes shall be used.
  + OK.
* The New box shall offer creation from scratch and out of a template – a database file in the special template folder. Each template should also provide a short description of the use case it covers.
  + OK. Templates must be provided.

## Role Oriented Configuration

The central configuration entity is Role. A role can be configured independently and then just be assigned to a node. Role is a logical entity that the user can easily imagine and work with, for example Master, Slave, Redundant Peer etc. On the top level a role requires a set of components. OK

In analogy with database tables where table attributes describe “functionality” of the given entity, components describe functionality of a role. Table relationships describe associations between the given entities. Associations between roles are data mappings that essentially map lists of source (output) and target (input) parameters. However roles can participate in associations in two slightly different ways.

* If the parameter list contains state parameters the role participates directly
* If the parameter list contains function parameters the role participates in context of a specific function instance (instance of the component function that owns these parameters). So here function instances provide a kind of additional “dimension” of the association.

In both cases every parameter mapped by a role must belong to a component related to this role.

The details of this must be discussed.

### Role UIC

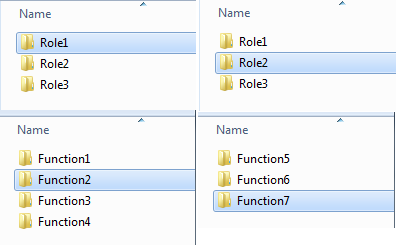
The role detail form shall only contain the role name and the list of related components and the list of related nodes. The relationship to the table “instance role” is not meaningful here. This relationship will be indirectly manipulated in the detail form of the data mapping UIC.

The Role Details screen already shows components. Roles could be added. What about Groups?

### Data Mapping UIC

The master view shall display the source role name and source instance (name:id) pair as well as destination role name and destination instance (name:id) pair.

The detail form shall contain the following subform:



Here a data mapping is configured between Role1:Function2 and Role2:Function7.

At the click onto a role the list of all instances indirectly belonging to this role (via the component relationship) shall be displayed. As said above, an instance must not be selected for a data mapping if the corresponding mapping side refers to a state parameter list.

Additionally the individual attributes of the data mapping (execution mode etc.) shall be displayed here as well as the list of individual parameter mappings.

### Parameter Mapping UIC

Parameter Mapping is an entity semantically fully owned by a data mapping. The master list simply presents names of the source and destination parameters. The detail form contains a subform similar to that of DataMapping UIC –a list at the left (source) and at the right (destination) side with the corresponding parameters selected. The edit mode of the respective list displays the list of the corresponding function or state parameters allowed by the current data mapping.

### Node UIC

As mentioned above, nodes are not much more that a combination of roles. However to specify the exact role configuration, a transparent concept of groups is introduced. Role relationships aka data mappings are only possible inside the same group. Since group is a pretty low-level and not easily understandable concept it should be hidden from the user.

Node master list is trivial.

Node detail form contains node attributes, the role relationship list and at the right side of it the list displaying the peer nodes accessible through the given role.

So if we select a role, then all nodes reachable from our node by means of this role are displayed. By activating the edit mode of this list we can add and remove nodes as desired. Adding nodes means we put them into the current group, removing them from the list takes them out of the group.

# Business Logic

The SQL data model can’t fully reflect the integrity and business rule requirements on the data objects in the system. It is assumed that the ORM layer takes over this responsibility and prohibits integrity violations on all levels including the lowest one (foreign key, check and not null constraints). This is because it is difficult to give the user reasonable feedback if the error is first detected by the database itself.

## Elementary integrity rules

### Not NULL constraint

Fields declared as NOT NULL in the database shall be checked by ORM for being non-empty.

### CHECK constraint

Check constraints (compliance of a value to a Boolean condition based on it) shall be imposed by ORM

### Unique Key constraint

This constraint ensures that a given set of values is present in the table only once. Unique keys on ordinary values are given in the model. However for brevity no such keys are currently defined for cross tables which are:

* Component Dependency (Component, Dependency)
* Component Role (Component, Role)
* Node Role (Node, Role, Group)
* Instance Role (Function Instance, Role)
* List Parameter (List, Parameter)
* Parameter Mapping (Mapping, Source, Destination, Transformation)
* Function Parameter (Component Function, Parameter)
* Data Mapping (Source, Destination, Mapping Mode, Key, Output Trigger)

The combination of values given in the parentheses shall be unique for the given table.

### Foreign Key constraint

This constraint ensures in the database that if table A references another one B by means of set of columns C then C contains values which correspond to some available primary key in the table B. This rule should also be imposed by ORM. It is not difficult to impose at the creation of a reference as database relationships are usually presented by object references in the corresponding language (here C#). It only has to be ensured that objects which are referred to are persisted before the references to them are persisted. It is more difficult to impose the rule at the deletion of the referenced object as in such a case all “users” of this object get effectively invalidated. In our case this means that deletion of the object should either be prohibited (RESTRICT propagation rule) or cause the deletion of all referring objects recursively (CASCADE propagation rule). Pls. provide a list of affected entities so that we can define what individual behaviour is appropriate in what case.

## Advanced integrity rules

### Non-cyclic component dependencies

An entity of table Component can be dependent on other entities from this table (a to-many relationship given by the cross-table Component Dependency). This reflects the fact that programming code contained in a component uses code from other components. By knowing about these dependencies the system can determine the right order of loading components into memory. However for this to work component dependencies may not build cycles. A cycle for example is present if component A depends on component B, B depends on C and C in turn depends on A. If such a cycle is present the system cannot decide what component to load first which leads to a system failure at the startup.

The software build process ensures there are no true cycles between software components but the user can introduce cycles in the database by entering incorrect dependency values. ORM shall ensure that no cycles are present by applying a cycle finding algorithm in directed graphs (i.e. http://stackoverflow.com/questions/261573/best-algorithm-for-detecting-cycles-in-a-directed-graph).

### Proper component layering

The system consists of 4 different layers given by the Layer enumeration table. Layers restrict component dependencies to be allowed only inside the same layer and one directly beneath it. So if a component A belongs to Layer Foundation, it can only be dependent on components in the Foundation and System layers. Other dependencies are not allowed.

### Unique parameter names

Parameters can be of two types – Function Parameters and Component States. The former describe the set of values the code of a function expects as it gets invoked. The latter represent the global state of the given component on a node. In the database both types are “derived” from the table Parameter as they share many common attributes. Name is one of such shared attributes. However the given Component Function cannot have multiple Function Parameter entities with the same name. So a check has to be made that within the set of Function Parameter entities referring to the same Component Function all names as found in the corresponding Parameter entries are unique. Similarly, a Component can never have multiple Component State entities with the same name. So a check has to be made that within the set of Component State entities referring to the same Component Function all names as found in the corresponding Parameter entries are unique.

# Glossary

|  |  |
| --- | --- |
| Term | Description |
| SuD | System under Development or System under Discussion. Refers to a software system in a given context (e.g. the context of a use case). |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# References

|  |  |
| --- | --- |
| Number | Description |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# Change History

|  |  |  |  |
| --- | --- | --- | --- |
| Rev. | Chapter | Description | Date  Name |
| - | All | Initial revision | 2010-09-14 Max Mustermann |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |