How to turn spreadsheets into object oriented models

Samuel Boutin, General Manager
Joe Matta, Project Manager

Knowledge Inside, 7C rue Jean Mermoz, 78000 Versailles, FRANCE
☎ : +33139027029
✉ : samuel.boutin@k-inside.com
✉ : joe.matta@k-inside.com

Abstract

It is very common to see engineers using spreadsheets (e.g. excel) as a small database. For instance, a data dictionary will be a spreadsheet with a column for the data identifier and then different columns for data attributes (e.g. data type, step, min and max value). An advanced use would be to add two columns for functions producing and consuming the data resulting in a cross-reference dictionary. Then many other applications exist related to project management, diversity management, requirements management and specialized fields topics. Spreadsheets generally capture a specialized field specification or transverse project information and all spreadsheets together can form most of a system specification.

A big part of engineering legacy data is provided in this format in many industries. This is partly due to the many functions of spreadsheets e.g. for billing, checking, simulate; using spreadsheet allows simple and effective checks for data integrity. A positive aspect is that spreadsheets contain already structured data w.r.t textual descriptions or drawings. But this does not help solving a major problem in systems engineering: keep all the different spreadsheets and specification documents aligned during the life of a project.

In this paper we investigate the possibility to upload a wide variety of spreadsheet descriptions into a synchronized system specification. To reach this goal, we proceed with the following steps:

- Provide a semantics to each spreadsheet
- Link this semantics to a DSL (Domain Specific Language) [2]
- Organize the semantics of all spreadsheets together
- Import the spreadsheet content in a unified object oriented model based on related DSL

As a result, we rise for many organizations the opportunity to leapfrog from a wide variety of semi-structured artefacts and specification documents to a synchronized multi-aspect model stored in a single database and then a straightforward introduction to model based design.

This concept extends more generally to the possibility to transform a database into an object oriented model and conversely. We also discuss how this relates to UML profiles, BMPN, and DSL initiatives.

1. Introduction

It is very common to see people (not only engineers) using spreadsheets (e.g. excel) as a small database. In Figure 1, we show the typical organization of such a spreadsheet. Usually, the first row is filled with the “titles” of the columns, and the other rows contain “instances” or the first row.

	A	B	C	D
1	Country	City	Is Capital	Population
2	France	Paris	true	10500000
3	France	Marseilles	false	1470000
4	France	Lyon	false	1470000
5	USA	New York	false	22000000
6	USA	Washington	true	8350000
7	USA	Chicago	false	9750000
8	Japan	Tokyo	true	34300000
9	India	Delhi	true	23300000
10	China			

Figure 1

Most of the time, the meaning of each first row item is not specified and is intended to be rather obvious according to the spreadsheet author. Also, the relationship between the different first row items is not specified. For instance in Figure 1, First row items “Country”, “City”, “is Capital” and “Population” are completely obvious because the example comes from a field of common knowledge to almost any educated person. Also, the relationship between these items is clear even if it is implicit. For instance the City is clearly belonging to the corresponding Country and “Is Capital” is certainly an assessment about whether the City is the capital or not of the related Country. It may be clear also that the population is the population of the city or the country. To assess all these implicit assertions, the reader can rely on its own knowledge, e.g. I know Marseilles is in France! Also, the related population is certainly the population of Marseilles because I know that the population of France is rather 60 Million.

Of course, if such an excel file is written in a specialized field, where we do not have sound references, the understanding of the meaning of the first row items will be very unclear so it makes sense to propose a language allowing specifying a “semantics” of the first row items and their relationships.

Also, from the example in Figure 1, it is quite obvious that a Country will contain many cities (or instances of City) and that “is Capital” and “Population” are properties of cities, something also called attribute in OO model theory. So there should exist some relationship between our spreadsheet semantics model and more classical modeling frameworks like UML, BPMN, DSL (Domain Specific Languages)…

On one hand, excel spreadsheet contain many legacy data. On the other hand modeling frameworks offer a graphical representation of systems and are the basis for Model Based Design. Establishing a clear link between both seems very attractive. So in this paper we investigate at the same time a language for the spreadsheet semantics specification and the translation into an object oriented modeling language.

In section 2, we address the language for specifying spreadsheet semantics, in section 3 we address the support for a translation into an object oriented model, in section 4 we propose use cases for the translation, in section 5 we discuss the positioning of this work w.r.t different kind of OMG frameworks. Finally, in section 6 we conclude and elicit further opportunities.

2. Meta-language for spreadsheets

If we come back to the example in Figure 1 in the introduction, the objective of this section is to propose a language to capture the different possible relationships between columns of an excel file. We will call the result a spreadsheet meta-model language.

To capture the possible relationship between two columns, it should be possible to specify at first that:

1. A column is included in another column, e.g. for Cities and Countries in Figure 1
2. A column is a produced or consumed flow from another column, e.g. a set of data produced by a function
3. A column is an attribute of another column, e.g. the population for a City
4. A column is related to another column, e.g. two countries may have a neighborhood.

In Figure 2 we propose a graphical language to capture three (3) possible relations we keep at this stage.

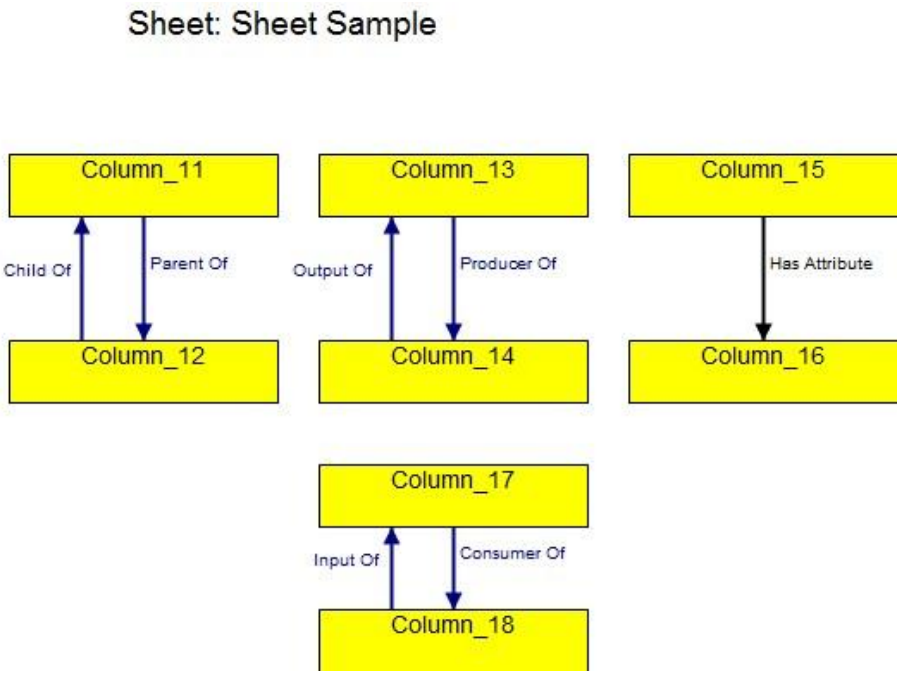

Figure 2

Columns are modeled as yellow boxes and relations between columns are modeled as flows between the boxes.

The inclusion relationship is specified by two flows: “Parent Of” and “Child Of” depending on the point of view we adopt: either container or contained object.

The flow relationship is specified by four possible flows “Producer Of, Consumer Of, Input Of, Output Of” depending of the point of view: from the flow itself or from the container and then depending on the orientation specified.

The attribute relationship is specified as one “Has Attribute” flow. Even if this not obvious at first, this diversity of representation is necessary for a practical implementation because the same column may be replicated in different sheets and we can say at this stage that the way it relates to other columns may vary.

Also, it seems reasonable to minimize the complexity of the spreadsheet meta-model language. So for instance, the interpretation of a relationship (fourth rule) may be interpreted as the existence of a link and the link itself may be interpreted as a flow between two columns. That flow would be then a virtual column. So the fourth rule can be implemented with the three others.

Finally, something is still missing at this stage: where will we import all the data? If we consider the sheet in Figure 1, all the countries can exist if we define a father object “World”

for instance. The father object may be more simply “Project”. It is not obvious when reading a spreadsheet to say what object will be at the top and what will not be. Anyway this cannot be determined automatically and we need to define a “Root” for the spreadsheet meta model. The root will tell us what columns shall be imported at the top and will also help a practical implementation.

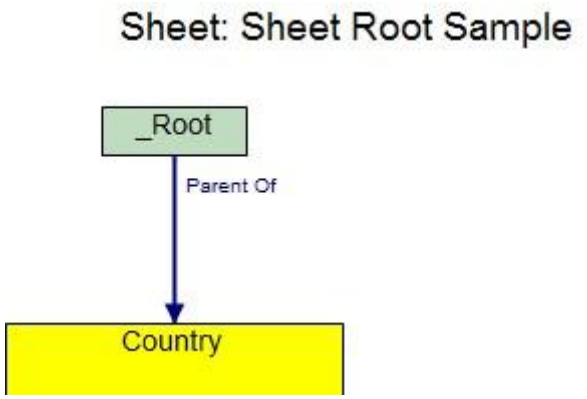

Figure 3

In Figure 3, you will find the description of a root object and how it is linked to a particular column. An attribute link would not make sense, but any link implying a hierarchical relation between root and a column may be allowed.

3. Transformation of spreadsheet in an Object oriented model

Assuming the notions of aggregation, flow and attributes are supported in our target meta-model, it is then possible to specify the translation from a spreadsheet to a model.

To do that, we will extend our spreadsheet meta-model description language to support the target meta-model description.

First we propose in Figure 4 a simple notation [3] for an object oriented meta-model supporting aggregation, flows and attributes.
It’s a simple tree like representation, in Figure 4:

1. « Country » is a class
2. « City » is a class
3. indentation means hierarchy and may be understood as Set theoretic inclusion
4. Each class is defined with a set of attributes
5. « is capital » and « population » are attributes

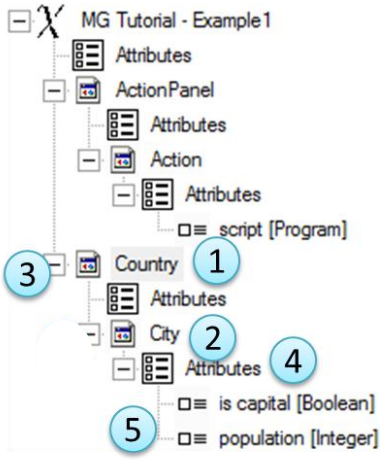

Figure 4

Although, flows are normally not first class citizen classes in object oriented languages, there is no contradiction between a functional and object oriented approach and clearly, this fits perfectly with the excel import in this paper. For arguments why this is a sound choice, consider for instance that objective C the Apple platform language is message based relying on Alan Kay’s work [1].

On the other hand, for the translation to be possible, we need to specify additional information for an excel column interpretation. In Figure 5, we propose an extension of the graphical representation of columns by adding additional information to a column representation (the yellow boxes) in the spreadsheet meta-model:
First, *columnName* is the name of the column shall be an attribute of the representation of a column. It should be noticed that the same identifier could be used for two columns in a spreadsheet. This shall be forbidden in one sheet, but we should not prevent it in different sheets. So the identifier of the imported column in the spreadsheet

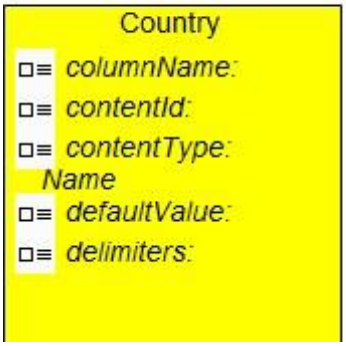

Figure 5

meta-model should be independent of the column identifier! This justifies the existence of the *columnName* attribute. The attribute *contentType* specifies whether the column will correspond to some class or some attribute in the target meta-model. The attribute *contentID* is the identifier of the attribute in the target meta-model and this name could be different from the one used in the source spreadsheet. Finally, *defaultValue* is necessary in the case of importing an empty cell and *delimiters* allow importing a list of objects from one cell (e.g. you get 3 objects or values, “a;b;c” in one cell with delimiter “;”).

Given this new information, we can nearly specify a transformation of an excel sheet into an object oriented model. The figure 6 represents our first description of the spreadsheet metamodel for Figure 1 example and its translation toward meta-model of Figure 4:

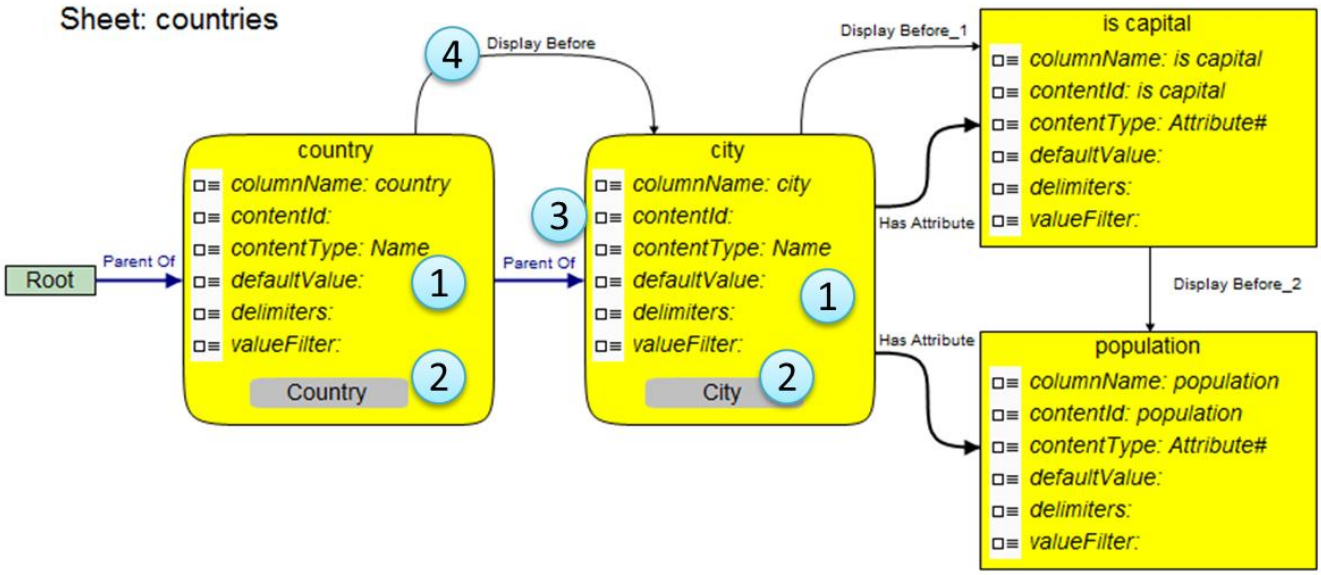

Figure 6

1. Yellow boxes correspond to excel columns
2. Grey boxes correspond to target classes in target class diagram
3. Some arrows correspond to rules of the target metamodel, e.g. « Parent Of » means hierarchy
4. Other arrows correspond to order of column in spreadsheet, e.g. « Display Before » models the order of columns in spreadsheet. This is useful in case of generation of spreadsheet from a model.

We will call “Rule Model” the spreadsheet meta-model with additional information allowing specifying its translation toward an object oriented meta-model.

Once the translation specification of the model is specified, all the necessary information to build a “rule file” are available. The rule file will be used to read each pair of excel cells and translate them according to the rule model. For

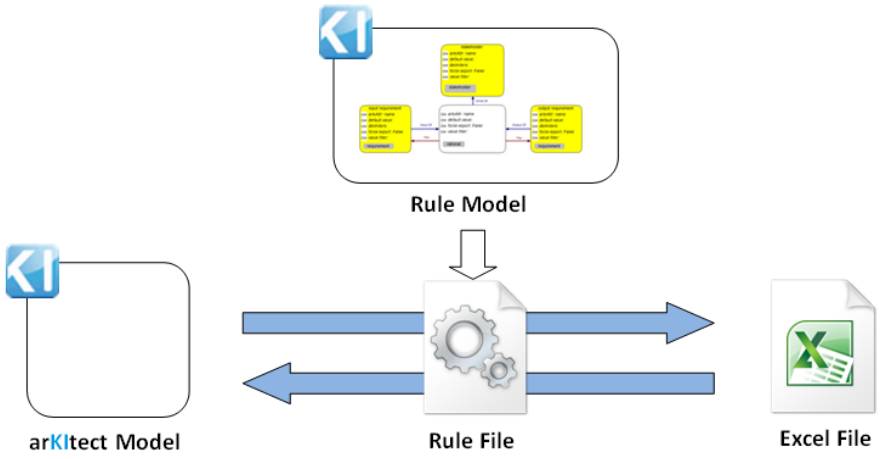

Figure 7

instance, if two excel cells correspond to an object and its attribute in the Rule model assuming the object is already translated, the translation of its attribute in a model is possible. This mechanism has been implemented successfully in the arKItect commercial tool. arKItect is an object oriented meta-modeler that can be configured to produce domain specific tools.
According to the import process we describe, it is very clear that the Rule Model has to be well funded meaning it shall be possible to position any column w.r.t the Root object. This is the reason why a diversity of representation is defined in Figure 2.

4. Use Cases of spreadsheet translation

Import of hierarchy of objects

If we consider the import of spreadsheet in Figure 1 using rule model of Figure 6, then we get the model in Figure 8.
Objects (or instances) correspond to boxes and you can see that some boxes are included in other boxes (e.g. Lyon(1) is included in France(3)).

Object (Class « Country » instance) France is expanded so that it is possible to see Class « City » instances membership.
For each instance of Class « City », e.g. « Lyon, an object is created in the modeler and the attributes « is capital » and « population » are filed in.

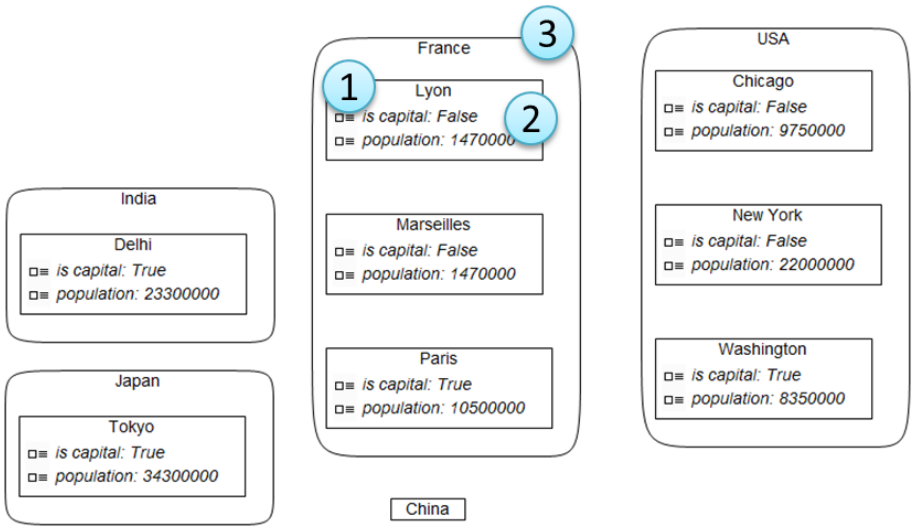

Figure 8

So we have now performed our first translation from a spreadsheet to a model and will present now different use cases of import and export. The topic addressed (countries, town, population) is far from any practical engineering aspect to avoid any misunderstanding about the examples and to show that all the mechanisms we experiment here are domain independent. However, once a meta model includes hierarchy of classes and flows between classes, it can be fairly considered as a DSL and supports system architecture modeling.

Exporting and updating

Note that to some extent, the translation from spreadsheet meta-model to an object oriented model can work in both directions. This is rarely an isomorphism because for example, empty cells during the import may have various interpretations.

	A	B	C	D
1	Country	City	Is Capital	Population
2	France	Paris	true	10500000
3	France	Marseilles	false	1500000
4	China	Beijing	true	16000000
5	Korea	Seoul	true	25100000

Figure 9

To update the target model with new data, you simply need to perform another import. The update creates new objects found in the spreadsheet and update modified attribute values. For example spreadsheet in Figure 9 will update the population of *Marseilles* and create the following new objects: *Beijing*, *Korea*, *Seoul*.

However, it is much more difficult to develop a mechanism for removing objects or relations between objects during an update. The right manner for such synchronization is certainly to perform difference and merge operations in the target model. Of course this requires support of the modeling tool for such operations.

Handling Recursive Types

Another use case is when a spreadsheet is used to specify a hierarchy of objects. Figure 10 is a sample of such use.

	A	B
1	System Parent	System Child
2	System 1	System 1.1
3	System 1	System 1.2
4	System 1	System 1.3
5	System 2	System 2.1
6	System 2.1	System 2.1.1
7	System 2.1.1	System 2.1.1.1

Figure 10

In such cases, one of the difficulties is to specify the root object that will contain the highest level of system (e.g. system 1 and 2 here). We can assume another import already created these objects.
In that case, and if the target object meta-model supports recursive types, it is possible to import a hierarchy with unknown depth. Indeed there is no need to specify how many levels of hierarchy are described in the spreadsheet

In that case, we see that in the rule model, the class “System” is used two times to interpret different columns, see Figure 11 that contains a minimal representation of both the target object meta-model and the spreadsheet meta-model (or Rule model)

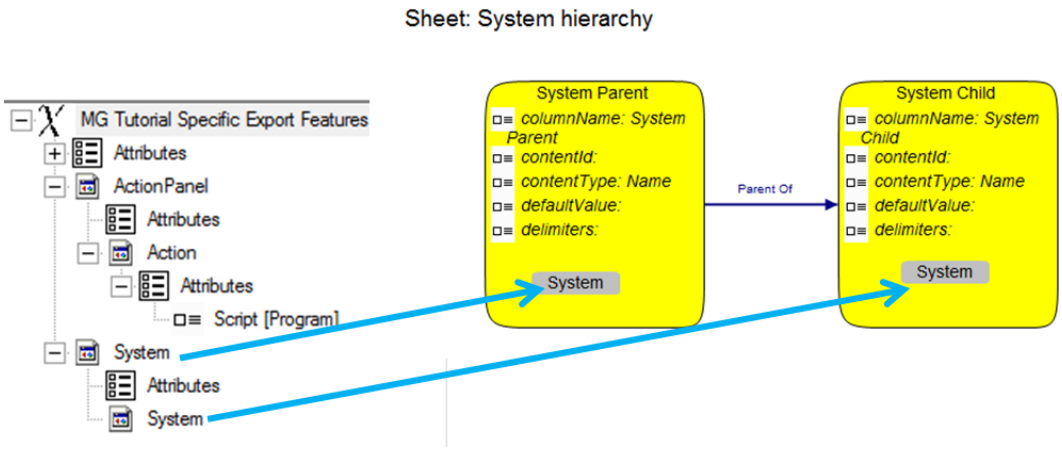

Figure 11

Importing: with the configuration of the Rule Model above, the high level systems must exist in your target project to import the hierarchy
First create the high level system in your target model. In our example the high level System objects are System 1 and System 2.

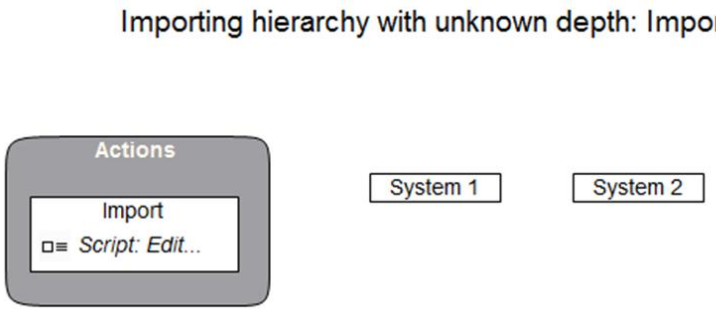

Figure 12

Then performing an import, you should get the following result (the *System* objects have been expanded to display the whole hierarchy in Figure 13.

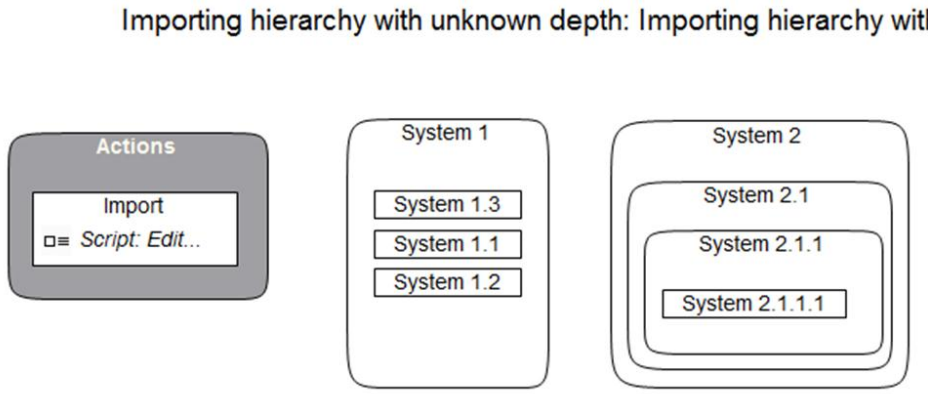

Figure 13

Complex case with flows:

Let's now consider a more complex spreadsheet in which we will interpret a column (namely "people flow" by a flow or message) in Figure 14. We display here only the first two lines of a complex excel sheet.

	A	B	C	D	E	F	G
1	continent	country of departure	town of departure	people flow	number of people	town of arrival	country of arrival
2	Asia	Lebanon	Beirut	People movement flow_2	15000	Bangalore	India
3	Asia	India	New Delhi	People movement flow_1	1000000	Beirut	Lebanon

Figure 14

The rule model for the import is displayed in Figure 15. Note that the column "people flow" is translated into a flow type named "people movement flow". Note also how the column "people flow" is linked to Column "Town of departure" by a "Producer of" link and to Column "Town of arrival" by a "Input of" link. This is the way a column can be interpreted as a flow. The target meta-model is not displayed now because it can be fully deduced from Figure 15. For instance, type "People movement flow" shall be a flow type under type "Town".

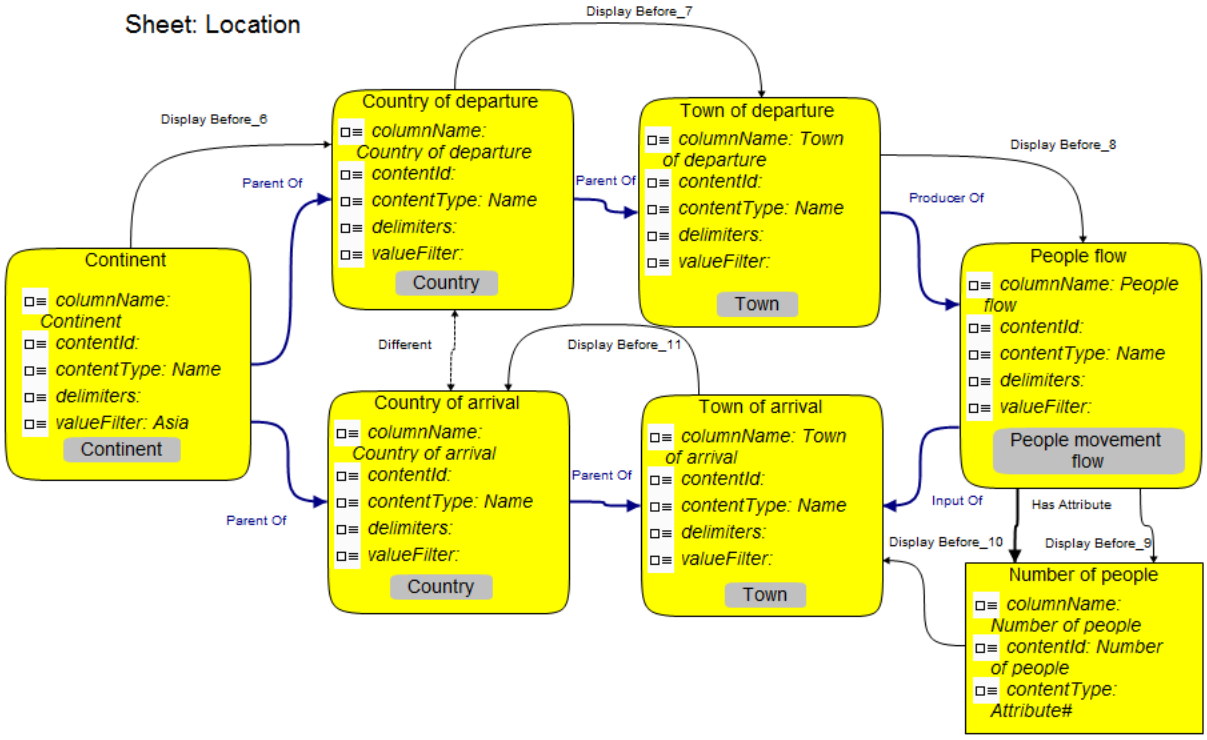

Figure 15

Then the resulting object model imported from sheet in Figure 14 would then look like the model in Figure 16. Look carefully and you will see the two flows described in Figure 14.

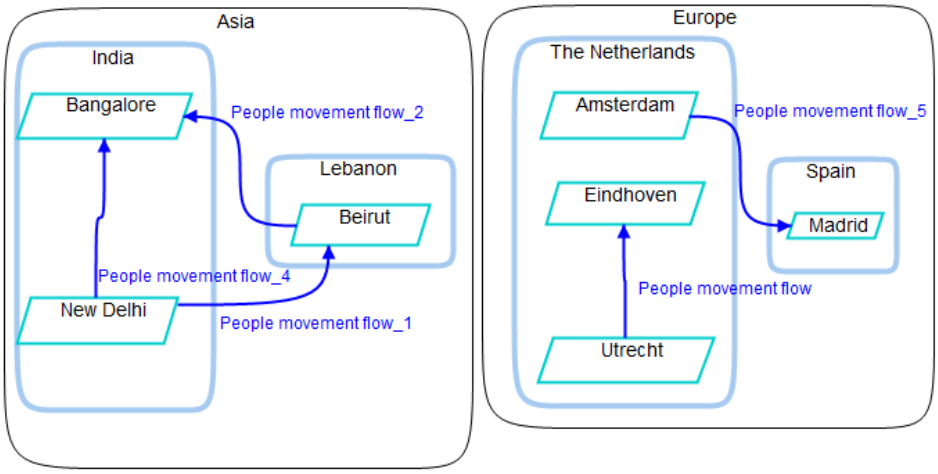

Figure 16

Notion of Key: case where object identification is not obvious

A very important difference between a spreadsheet and a model is usually that in a spreadsheet, the notion of cell is really central and the content of the cell is not a reference in itself. But in a model, an object is a reference and the object may be referred to at different places. From this we deduce in the important concept of Key allowing detecting that two identical identifiers in two cells of a spreadsheet eventually correspond to different objects. Example of a spreadsheet where this case happens is displayed in Figure 17.

	A	B
1	Championship	Group
2	World Rugby Cup	Group A
3	World Rugby Cup	Group B
4	World Rugby Veteran Cup	Group A
5	World Rugby Veteran Cup	Group C

Figure 17

Here we see that “Group A” occurs two times and the two occurrences are not equal because they refer to different competitions. So we need to define a new kind of relationship in the spreadsheet meta-model of Figure 18: the Key relationship.

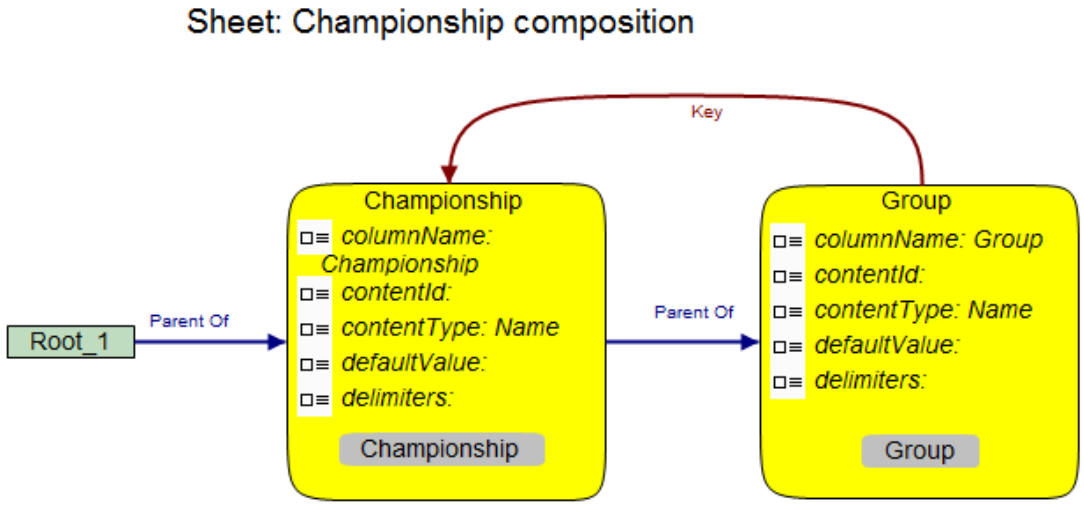

Figure 18

The red link “Key” specifies that two groups with same name are in fact different if the belong to different Championship.
The target object model should look like Root/Championship/Group.
The resulting import would then be as in Figure 19.

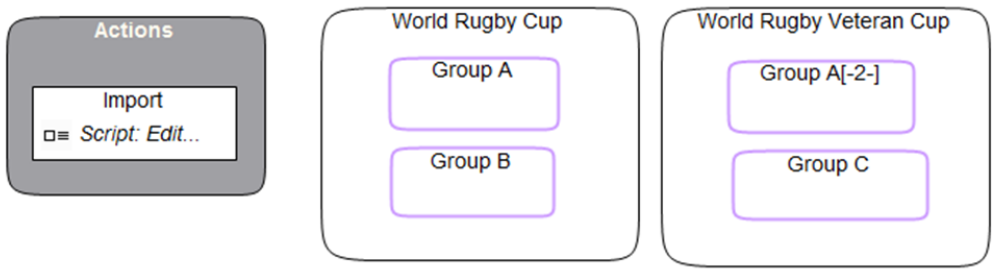

Figure 19

As can be seen, the two occurrences of “Group A” have been imported as two different objects. So it is necessary to rely on some automated naming mechanism to support such cases.
There are still many use cases to explain in order to get an overview of the Rule models power and implementation but the previous examples provide a flavor of the whole feature.

5. Relationship to OO frameworks.

Modeling tools based on OMG concepts [5] have specialized on different markets along the latest decade. They have evolved from software development to support various standards for a wide variety of applicative domains: MOF, UML, MDA, UML2, BPMN, SysML SoaML,…). Although UML and similar tools are still mainly use for software development, there is a strong faith among their supporters that this technology will be able to rule all the domains of system design. However, even for software design one can find very strong opponents [1], and it's rather interesting to note that the technology we present in this paper is "message based" and closer to the Smalltalk and Objective C (The Apple apps programming language!) community than to the OO OMG approach.

In the paper, the mechanisms we have proposed are a complementary approach to usual OO modeling. Rather than trying to define a very generic and multi-purpose meta-model framework, we propose to be rather data centric, start from existing database description, and add semantics to all these parts and propose integration mechanisms using the rule model as described in section 2 and 3. The advantage of such an approach is that it is much more attractive to creative people and engineers that do not want to change their data organization. Also, the advantage is that switching from database point of view to model visualization would immediately bring value to end users without needing complex model transformation or setting up a model from scratch.

The approach described in this paper allows setting up a complex DSL from an existing database with the advantage that the DSL will reuse the glossary of the end user and will be immediately fit for purpose.

6. Conclusion

In this paper we propose a general mechanism to provide a meta-model for spreadsheets. Each column of the spreadsheet will be interpreted as a key identifier for an object or an attribute of an object or a parent of an object (in a hierarchy) or a flow or simply a link.

This mechanism can be implemented to build efficiently Domain Specific Language and benefit from the power of database, spreadsheet and model based design at once.

Also, the spreadsheet meta-model allows providing a semantics that is most of the time implicit and not documented. It is also an opportunity to set up the relationship between different spreadsheets when possible.

We think this approach is very valuable for legacy data import and is a very efficient way to introduce seamlessly model based design in engineering teams.

References

[1] Alan Kay : *Steps toward the reinvention of programming ;* NSF Grant position paper, 2006

[2] Benoît Langlois *: Toward Families of QVT DSL and Tool ;* OOPSLA, 2006

[3] Knowledge_Inside *: arKItect documentation ;* 2011, Récupéré sur https://support.k-inside.com/display/ARKI11/arKItect+1.1.x+documentation+home

[4] Manfred Broy : *Seamless model-based development: From isolated tools to integrated model engineering environments ;* Proceedings of the IEEE, pp. 526 - 545, volume 98, n° 4, 2010

[5] OMG : *Meta Object Facility ;* http://www.omg.org/mof/