Analysis and Development of a Prototype Based on Media Wiki and Semantic Media Wiki Integrated with a Design Modeling Tool that Allows Modeling of Organizations Based on Demo

MASTER DISSERTATION

Duarte Nuno Freitas Pinto
MASTER IN INFORMATICS ENGINEERING

UNIVERSIDADE da MADEIRA
A Nossa Universidade
www.uma.pt

September | 2012

Analysis an Development of a Prototype Based on Media Wiki and Semantic Media Wiki Integrated with a Design Modeling that Allows Modeling of Organizations Based on Demo

MASTER DISSERTATION

Duarte Nuno Freitas Pinto
MASTER IN INFORMATICS ENGINEERING

ORIENTAÇÃO
David Sardinha Andrade de Aveiro

Abstract

In a world where organizations are ever more complex the need for the knowledge of the organizational self is a growing necessity.

The DEMO methodology sets a goal in achieving the specification of the organizational self capturing the essence of the organization in way independent of its implementation and also coherent, consistent, complete, modular and objective.

But having such organization self notion is of little meaning if this notion is not shared by the organization actors. To achieve this goal in a society that has grown attached to technology and where time is of utmost importance, using a tool such as a semantic Wikipedia may be the perfect way of making the information accessible.

However, to establish DEMO methodology in such platform there is a need to create bridges between its modeling components and semantic Wikipedia.

It's in that aspect that our thesis focuses, trying to establish and implement, using a study case, the principles of a way of transforming the DEMO methodology diagrams in comprehensive pages on semantic Wikipedia but keeping them as abstract as possible to allow expansibility and generalization to all diagrams without losing any valuable information so that, if that is the wish, those diagrams may be recreated from the semantic pages and make this process a full cycle.

Keywords: DEMO, Semantic MediaWiki, Organizational Modeling, UVMMD

Resumo

Num mundo em que as organizações são cada vez mais complexas a existência do conhecimento do "ser" organizacional é uma necessidade crescente.

A metodologia DEMO propõe-se a conseguir esta especificação do "ser" organizacional capturando a essência da organização de forma independente da sua implementação e de maneira coerente, consistente, completa, modular e objetiva.

Mas de pouco serve deter a noção sobre o "ser" organizacional se esta não for partilhada pelos seus intervenientes. Para este fim, numa sociedade cada vez mais ligada à tecnologia em que o tempo é um bem essencial, a utilização de uma ferramenta como um Wikipedia Semântico pode ser o modo perfeito de divulgação.

Mas para conseguirmos estabelecer a metodologia DEMO em tal plataforma, é necessário estabelecer pontes entres os seus componentes e o Wikipedia Semântico.

É nesse aspeto que a nossa tese se foca, tentando estabelecer e implementar em um caso de estudo os princípios de uma forma de transformar os diagramas da metodologia DEMO em páginas compreensíveis no Wikipedia Semântico mas de forma o mais abstrata possível para permitir expansibilidade e generalização a todos os diagramas e, sem perdas de informação, para que, se tal for o desejo esses diagramas possam ser recriados a partir das páginas semânticas e assim tornar o processo num ciclo completo.

Palavras-chave: DEMO, Semantic MediaWiki, Modelação de Organizações, UVMMD

Index

Image Index

Table Index

1. Introduction

1.1. Motivation

This Project idea derives from the fact that most software development projects fail to accomplish the initial expectations of its users, and one of the identified causes for that to happen is the insufficient or inadequate knowledge of the organization's reality that lacks to be automated or supported by one information system.

In order to fill that gap, on the 90's a new discipline was born, Organizational Engineering, that gathers the concepts and methods of engineering and applies them to organizations to comprehend and represent the multiple tasks of that organization and also facilitate the analysis and the organizational change, regardless of the information system implementation.

On another front the semantic wiki's are tools of easy reach and easy to be used by anyone despite their area of knowledge. This tool makes possible to collect distributed and coherent information about the organizational knowledge in the shape of elements and semantic relations represented in models that are true to the organizational reality and allow the capture and following of their evolution and, also makes possible, a faster and more efficient way of developing a information system that supports that organization's reality.

And from these two points evolves this thesis objective, establish a bridge between a design modeling tool and semantic wiki to allow a semi-automated creation of wiki pages having as a starting point a diagram and avoid the data loss to allow the reversion of that process if desired.

1.2. Objectives

The objectives of this thesis is to do an analysis and develop a prototype based on Media Wiki and Semantic Media Wiki and its respective integration with a design modeling tool so it allows the modeling of organizations based on DEMO (Design and Engineering Methodology for Organizations). This process should happen without relevant data loss leaving the possibility to reverse the result back into the starting model.

To that goal there is the need to find a design modeling tool in which allows the implementation of DEMO diagrams and a way to export those models so they can be worked on and integrated in Media Wiki.

A careful analysis of such tool and the models in general is also a vital needed step to ensure that all the relevant information is kept and that the process is possible to revert.

To make the implementation functional but still as flexible as possible to allow expansibility and new sets of diagrams is another objective to be considered.

This flexible implementation and tool modeling components analyses are the major contributions in this thesis as together they should allow support and validation for future developments.

Overall, it should be possible for an organizational engineer to create models in the chosen designing tool and then easily upload it into wiki, leaving the information in those organizational models accessible to all members of the organization.

Those members should then be able to use a specific Wiki interface to see, propose and approve changes in the official organizational models.

1.3. Organization

In the second chapter of this thesis we will explain the Design and Engineering Methodology (DEMO) for Organizations and the basic principles needed to understand it. A few notions are required in order to understand the fundaments of DEMO, starting by the definition of Enterprise Ontology, Model Triangle and the World Ontology Specification Language (WOSL).

In this same chapter we will explore the DEMO Meta model, Media Wiki, Semantic Media Wiki (SMW) and all the components that we will need to use related to SMW.

Finally in the second chapter we will have a brief overlook on some of the diagram design tools available on the market and their ability to support DEMO diagrams and exporting them for a workable format that we could use to accomplish our objectives.

In the third chapter we dealt with the implementation and all the steps done in the process of developing a semi-automated way to create SMW pages for our DEMO diagrams keeping all the needed information for this process to be reverted and brought back to the diagram state.

We started this chapter with a case example of a library taken from J. Dietz book Enterprise Ontology. Then we have a brief approach to validation rules, and their possible relevance followed by some changes propositions to the DEMO Meta model.

Latter in chapter three we approached and exchangeable format to keep all the diagrams information followed by the library case implementations in Xemod, and Actor Transaction Diagram implementation in Dia.

Kivio was our first breakthrough in software that we could possibly work with. As so, in this chapter we approach Kivio's implementation of the stencils and diagrams with some more detail, and try our first resolution for a PHP parser to export the relevant data into an exchangeable format.

Still in this chapter after highlighting Kivio's flaws we focus our attention in Visio, were we develop two solutions, the first, while a viable solution was too problem oriented to this specific case, and the second, more abstract and allowing future changes in notations and involved variables. In this implementation there is also the need for a SMW solution for our web pages, and as such it's also approached in the chapter.

On the forth chapter we thought about that could be done in the future to better profit from this thesis, and how those steps could pass for the development of diagram rules in Visio, the generalization of the Visual Basic for Applications (VBA) code and the most important, completing the cycle to make the web pages back into a diagram.

In the fifth chapter we do a brief conclusion of our work, and contributions.

2. State of Art

2.1. Basic Notions

Sections 2.1.1, 2.1.2, 2.1.4 and 2.1.5 are mostly based and contain texts from J. Dietz book "*Enterprise Ontology: Theory and Methodology*". [1]

The first step of trying to do something new is to understand what already exists, and comprehend the basic notions that are going to be applied in the process. For instances since the objectives central piece is the Design and Engineering Methodology for Organizations (DEMO) the first obvious step in this case is to understand what are the objectives and concepts of this methodology and how is it applied. There was also the need to previously understand other linked concepts like enterprise ontology, model triangle, ontology of a world and world ontology specification language.

With those notions in mind it was then possible to concentrate on the tools to be used.

2.1.1. Enterprise Ontology

The first step to understand what is an Enterprise Ontology is to understand the meaning of Ontology. Ontology derives from two Greek words *onto* and *logia* the first meaning "*knowledge of the being"* and the second meaning "Study" and is a part of philosophy that studies the nature of being, existence, reality, and the basic categories of the being and their relations.[2]

In this particular case our being is the Enterprise, and as such Enterprise ontology can be defined *as "a formal and explicit specification of a shared conceptualization among a community of people of an enterprise (or a part of it)*." [1]

This Enterprise Ontology has to satisfy five quality requirements:

- Coherence, all the distinguished aspect models constitute a logical and integral conceptual model;
- Comprehensiveness, all the relevant matters are covered;
- Consistency, there are no irregularities or contradictions in the aspect models;
- Conciseness, no superfluous matters are contained in it;
- Essence, the aspect models are abstract from any implementation or realization issue, it only represents the essence of the enterprise.[1]

2.1.2. Model Triangle

Before we can understand the Model Triangle, we need to have two previous concepts, the notion of what is a system, and the notion of what is a model.

There are two different notions of a system, the teleological and the ontological. The teleological notion concerns its function and external behavior and is adequate for its use and control. And the ontological notion concerns its construction and operation and is necessary for building and changing systems. [1]

For the understanding of the Model Triangle we need the ontological notion of a system. In this notion, according to J. Dietz, "*something is a system if and only if it has the following properties:*

- *Composition: a set of elements of some category (physical, social, biological, etc.).*
- *Environment: a set of elements of the same category; the composition and the environment are disjoint.*
- *Production: the elements in the composition produce things (e.g., goods or services) that are delivered to the elements in the environment.*
- *Structure: a set of influence bonds among the elements in the composition, and between them and the elements in the environment*."[1]

The best definition of model to suite our needs is to quote Leo Apostel; *"Any subject using a system A that is neither directly nor indirectly interacting with a system B, to obtain information about the system B, is using A as a model for B."*[4]

With this definition we take that the notion of model is a notion of role, this is, something that is not inherently a model, but may be used as one. [1]

In this context modeling a system can be divided in three categories; concrete systems, symbolic systems and conceptual systems. The relations between these models can be explained by the model triangle represented in the image bellow. [1]

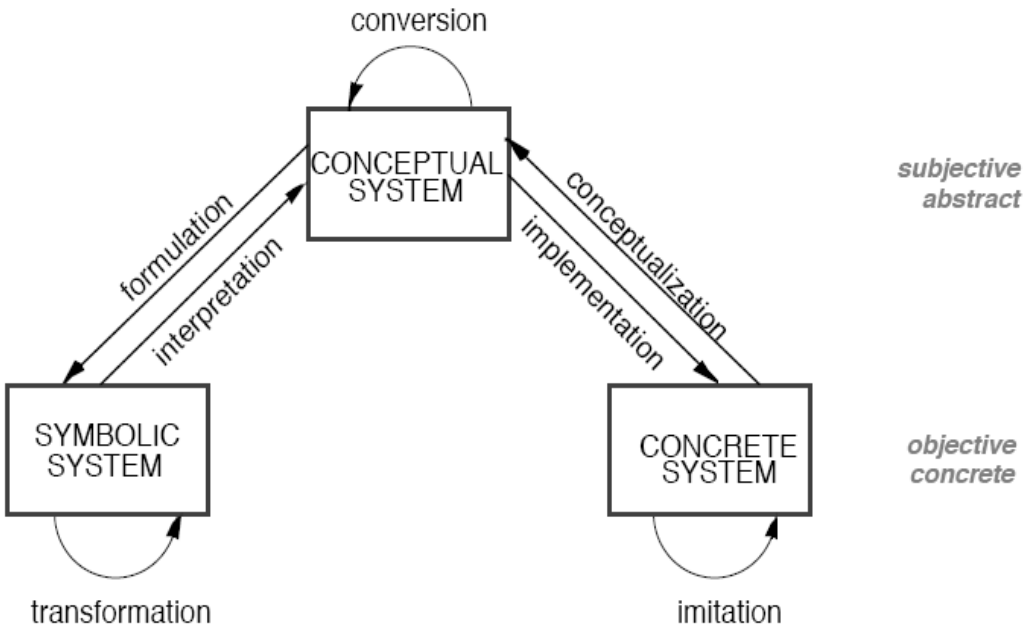


Image 1: Model Triangle [1]

To better explain the model triangle we will quote J. Dietz definition;

"For a convenient and unambiguous explanation of the model triangle, let us call an X-type system that is used as a model for some system an X-type model of that system. So, for example, a conceptual system that is used as a model of a car is just called a conceptual model of the car. Then, the next explanation holds:

A concrete model of a concrete system is called an imitation. Examples: a scale model of an airplane or a ship or any other concrete thing. The reason for building an imitation of a system is generally that it is easier, cheaper, less dangerous, etc., to study the model instead of the system itself.

A conceptual model of a concrete system is called a conceptualization. It plays a major role in all sciences. Examples: the geometrical sphere as a model for celestial bodies; the (feedback) control system as a model of biological or technical or managerial processes; the Process Model as the conceptualization of the (business) processes in an enterprise.

A concrete model of a conceptual system is called an implementation. Examples: the pyramids of Giza are an implementation of the geometric concept of pyramid (though, one might argue whether actually the reverse is true); James Watt's steam engine regulator as an implementation of the (feedback) control system; a business process as an implementation of the Process Model.

A conceptual model of a conceptual system is called a conversion. Examples: the algebraic concept of a circle (x2 + y2 = r2) is a conversion of its geometric concept (and vice versa); v = a.t is the algebraic conversion of the physical concept of the uniformly accelerated movement.

A symbolic model of a conceptual system is called a formulation. A symbolic system is expressed in some formal language. People often become so used to the notations in languages that they equate the expressions with the conceptual models they represent. A splendid example is the algebraic concept of a circle. We referred to it above by the equation x2 + y2 = r2, meaning the notion that is expressed by it, and not the notation. As the symbolic model of it, however, we mean the notation. Other examples: the Peano-Russell notation of a logical formula; the Actor Transaction notation of an Interaction Model.

A conceptual model of a symbolic system is called an interpretation. Since interpretation is the reverse of formulation, we refer to the examples given above. Other examples: educing the meaning of a data flow diagram; the deciphering of the Stone of Rosetta.

A symbolic model of a symbolic system is called a transformation. It is also often called translation. However, one should be cautious in doing that. Translation presupposes understanding of what is being written (or spoken); one goes back and forth interpretation and formalization, so to speak. By transformation we really mean trans-coding, e.g., from ASCII to EBCDIC, or from Morse to the Roman notation of letters.

One should keep in mind that a conceptual model is something in the mind. It is not identical to the symbolic system (sketch, diagram, etc.) in which it necessarily has to be formulated in order to be able to communicate the conceptual model to somebody else or to oneself at a later point in time." [1]

2.1.3. Ontology of a World

To fully grasp the ontology of a world we will use parts of a summary on that matter presented in *G.O.D. (Generation, Operationalization & Discontinuation) and Control (sub)organizations: a DEMO-based approach for continuous real-time management of organizational change caused by exceptions* by D. Aveiro. [7]

"The adopted notion of world is the discrete event world, with the assumption of a discrete linear time dimension. It means that the time difference between any two consecutive points in time is the same. This time difference is called the basic time unit. A world can formally be defined as a pair <C,B> with C being a set of objects, called the composition and B a set of facts, called the state base. The 'existence' of an object is assumed even if one cannot know anything about it. The number of objects in C is denumerable infinite; so, there will never be a shortage of objects: all things that are interesting or that may become interesting are already there.

A fact is a particular arrangement of one or more objects. Depending on the number of objects that are involved in a fact, one speaks of unary, binary, ternary, etc., facts. An example of a unary fact is that Beatrix is the Queen of the Netherlands. An example of a binary fact is that Willem Alexander is a son of Beatrix.

A fact is said to be current at a particular point in time if it is the case at that point in time. At any moment, a world is in some state. The state S at time t is defined as the set

of facts that are current on t, $S \subset B$. A state change is called a transition. A transition is defined as an ordered pair of states, e.g., T1 = <S1,S2> is the transition from the state S1 to the state S2. The occurrence of a transition is called an event. An event therefore can be defined as a pair <T1, t>, where T1 is a transition and t is a point in time. Consequently, a transition can take place repeatedly during the lifetime of a world; events however are unique: they occur only once. Often, one talks about event type instead of transition.

In order to understand profoundly what a state of a world is, and what a state transition is, two kinds of facts are distinguished: dependent and independent facts. A dependent fact is something that is the case and that will always be the case. For good reasons one can even say that it has always been the case. In other words, it is an inherent property of an object or an inherent relationship between objects. A independent fact is something that becomes the case as the effect of an act in the corresponding system.

Examples of dependent facts in the context of the library are the ones expressed in the next assertive sentences (in which the variables, represented by capital letters, are placeholders for object instances): "the author of book title T is A"; "the membership of loan L is M". On the one hand, the existence of these facts depends on the existence of the corresponding book title and loan respectively. On the other hand, dependent facts are timeless. For example, a particular book title has a particular author (or several authors). If this is the case at some point in time, it will forever be the case. One might even say that it has always been the case, that it was only not knowable before some point in time (namely before the book was written). This position is adopted for ease. A similar remark holds for every defined fact. A defined fact is determined by its definition. The specification of this definition is the only necessary and sufficient condition for the existence of the defined fact. This marks an important difference between a world and an information system about that world. For example, the age of a person (which is a defined fact) ontologically just exists at any moment; in the corresponding information system, it has to be derived, i.e., computed when it is needed.

Dependent facts are subject to existence laws. These laws require or prohibit the coexistence of facts (in the same state of a world). For example, if the (single) author of some book is "Ludwig Wittgenstein", it cannot also be "John Irving".

Contrary to a dependent fact, an independent fact is the result or the effect of an act. Examples of independent facts in the context of a library are the ones expressed in the next perfective sentences (the variables are again placeholders for object instances): "book title T has been published", "loan L has been started".

The becoming existent of an independent fact is an event. Before the occurrence of the event, it did not exist (i.e., it was not the case); after the occurrence it does exist (i.e., it is the case and it will forever be the case).

Events are subject to occurrence laws. These laws require or prohibit sequences of events in the course of time. For example, sometime after the occurrence of the event

"loan L has been started", the event "loan L has been ended" might occur, and, in between, several other events may have occurred, such as "the fine for loan L has been paid". Therefore, events can best be conceived as status changes of a thing.

The following is considered to be a precise definition of the ontology, or, more precisely, the ontological model of a world: the specification of its state space and its process space. Both are expressed in business rules. A clarification of each key term follows:

By state space is understood the set of allowed or lawful states. It is specified by means of the state base and the existence laws.

The state base is the set of dependent fact types of which instances may exist in a state of the world. The existence laws determine the inclusion or exclusion of the coexistence of facts.

By the process space is understood the set of allowed or lawful sequences of events. It is specified by the event base and the occurrence laws.

The event base is the set of event types of which instances may occur in the world. Every such instance has a time stamp, which is the occurrence time of the event. Existence laws and occurrence laws are expressed in business rules.

There are two possible shapes of business rules: declarative and imperative.

An example of a declarative rule, in a library, is: no more than 5 books can be borrowed at the same time under one membership. The same rule can be expressed imperatively in the next way: when a loan is requested under some membership if there are already 5 books in loan under this membership then decline the request else promise the request

It should be noted that both the declarative and the imperative formulation are rather informal, a choice done for the sake of simplicity. It is stated that the choice for one of the two kinds of formulation in practice is highly dependent on the (presumed) competences of the actors who are going to apply them and that the imperative shape is the most appropriate candidate for automation." [7]

2.1.4. World Ontology Specification Language

The World Ontology Specification Language (WOSL) is a language that was proposed by J. Dietz to specify the ontology of a world that unlike the other ontology languages it included the transition space of such world in addition to its state space. [6]

This Language is relevant to our work because it is used in both the DEMO methodology's state model but also to specify the DEMO Meta model that we will approach further in this thesis and latter propose some changes.

To understand the WOSL grammar we will still need two more notions that can be inferred by the ontology of a world provided in 2.1.3. The first is the notion of Statum;

something that has always been, is and always will be the case, as it has external influence. And the second is the notion of Factum; the result of an act, it comes to existence at a certain point in time and remains as such forever. [1]

2.1.4.1. WOSL grammar

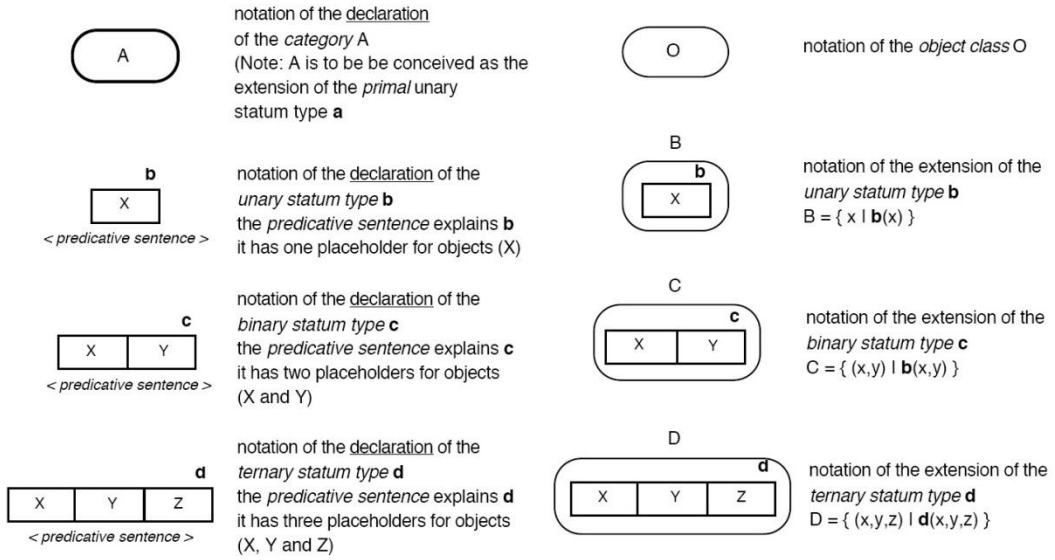


Image 2: Statum Type Declarations [1]

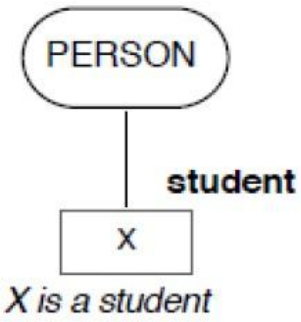


Image 3: Reference Law [1]

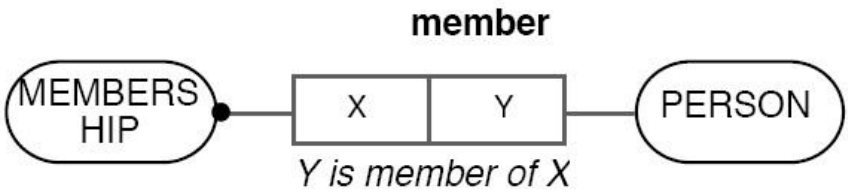


Image 4: Dependency Law [1]

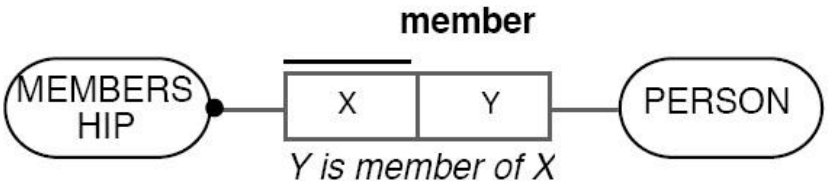


Image 5: Unicity Law [1]

2.1.5. Design and Engineering Methodology for Organizations

The Design and Engineering Methodology for Organizations (DEMO) is based on the Ψ - Theory about the operation of organizations. So, in order for us to understand DEMO, we are going to first explore the Ψ – Theory. [1]

Ψ Stands for PSI; Performance in Social Interaction and that is the paradigm in which this theory was founded. This theory is composed by four axioms and one theorem; [1]

- Operation axiom states that: Subjects in the organization perform two kinds of acts; production acts and coordination acts. A subject performing one of these acts is an actor and as such is performing an actor role.[1]

  By performing production acts they contribute to achieve the goal of the of the organization; [1]

  By performing coordination acts they comply with the commitments regarding the production acts; [1]

- Transaction axiom states that: Both coordination and production acts occur in a particular generic sociological pattern called transaction. [1]

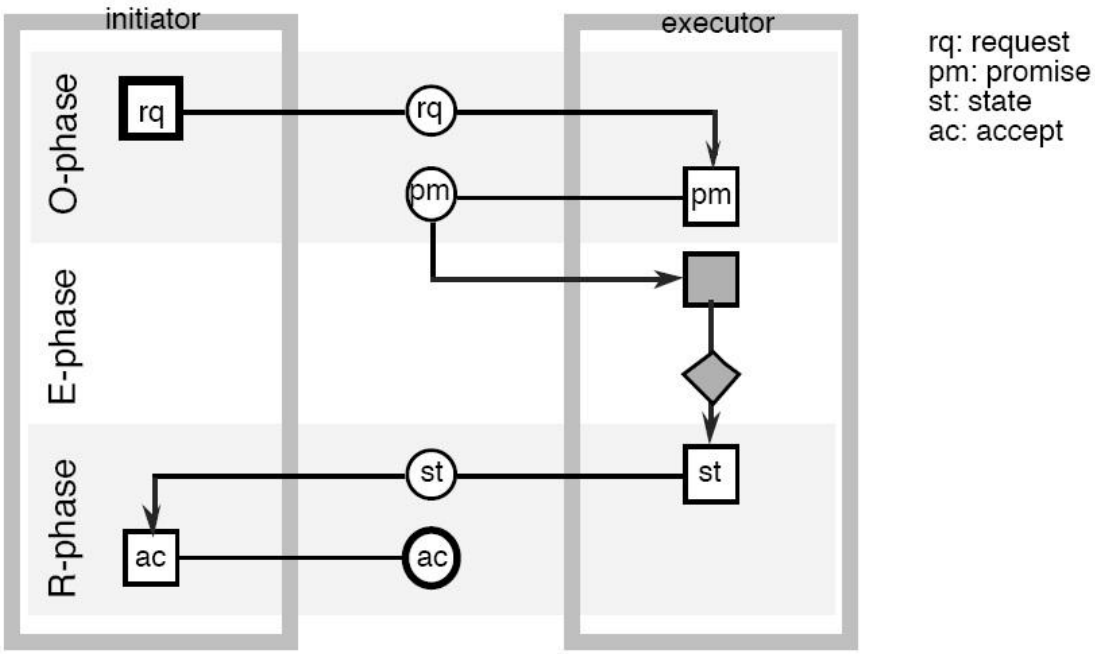


Image 6: Basic Transaction Pattern [1]

- Composition axiom states that: Every transaction is either: enclosed in another transaction; a self activating transaction or a costumer transaction. [1]

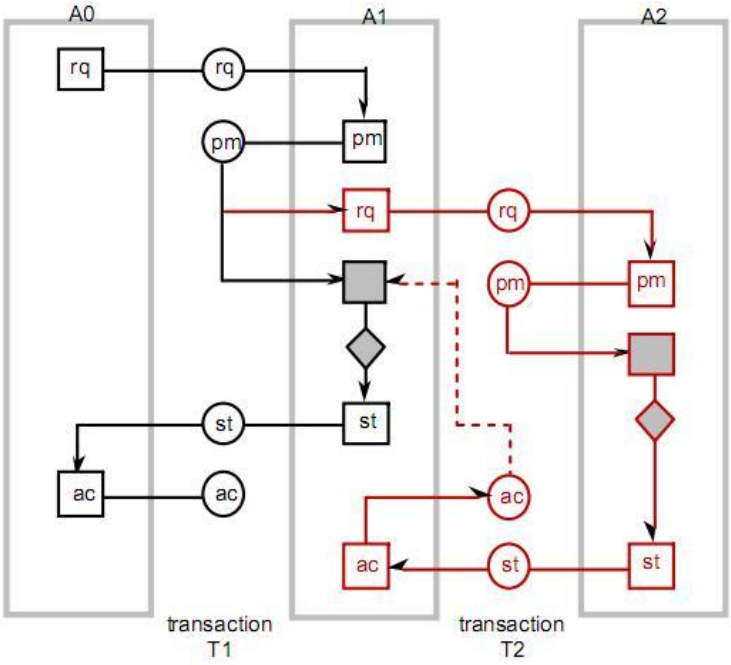


Image 7: Enclosed Transaction [3]

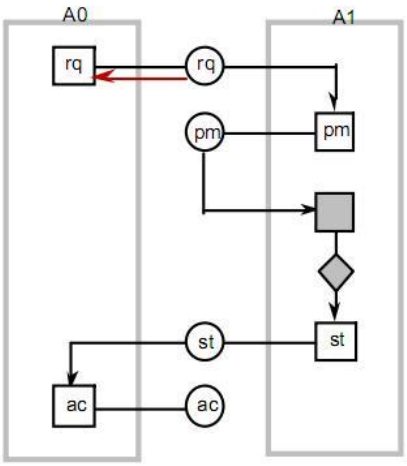


Image 8: Self Activating Transaction [3]

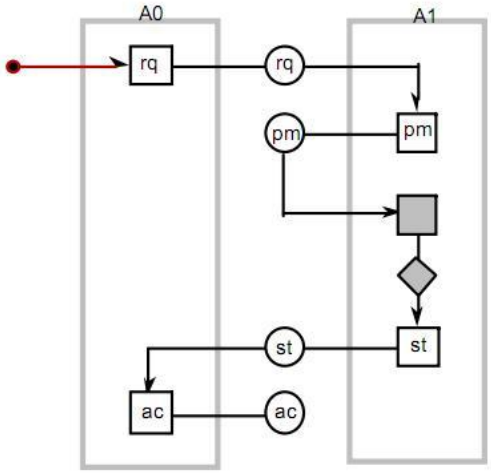


Image 9: Costumer Transaction [3]

Distinction axiom states that: There are three distinct human abilities that play a role in an actor operation; performa, informa and forma. [1]

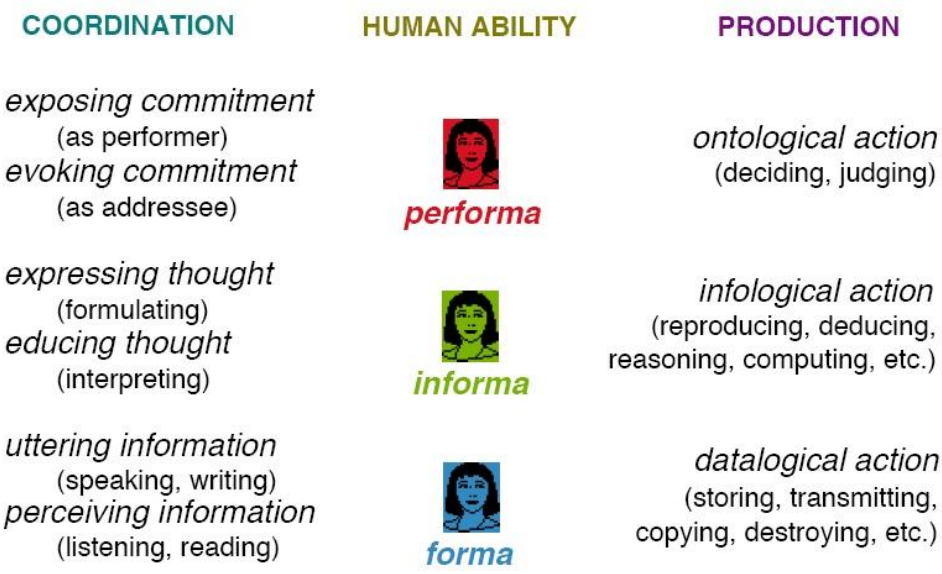


Image 10: Distinction Axiom Summary [1]

- Organization theorem states that: An enterprise organization is a heterogeneous system composed by the layered integration of three homogenous systems; the Document-Organization, the Intellect-Organization and the Business-Organization. The Business-Organization (B-Organization) is where the ontological production happens, and it is supported by the Intellect-Organization (I-Organization). The infological production happens in the I-organization. The Intellect-Organization is also supported by the Document-Organization (D-Organization) where the datalogical production happens. [1]

  The image bellow represents the organization theorem with the approximated rations of each production. This is ratio is approximately 1:4:7 respectively. [1]

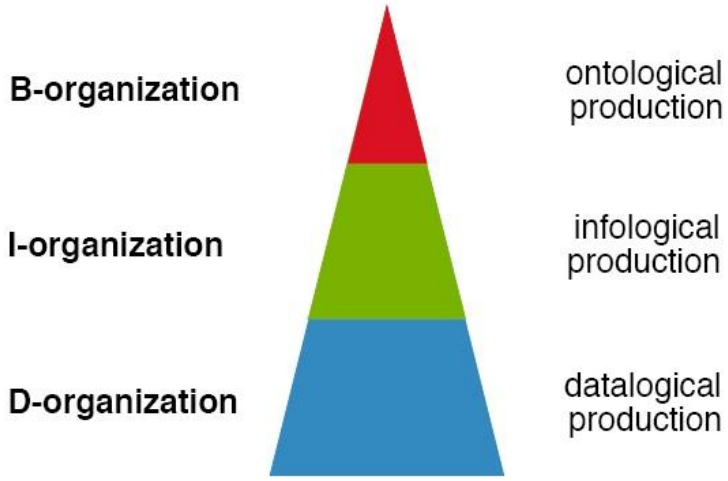


Image 11: Organization Theorem Representation [1]

With the Ψ – Theory explained we can say that DEMO is the practical demonstration of such theory. [1]

DEMO states that the ontological knowledge of an enterprise can be expressed in four aspect models in such way that is accessible and manageable. [1]

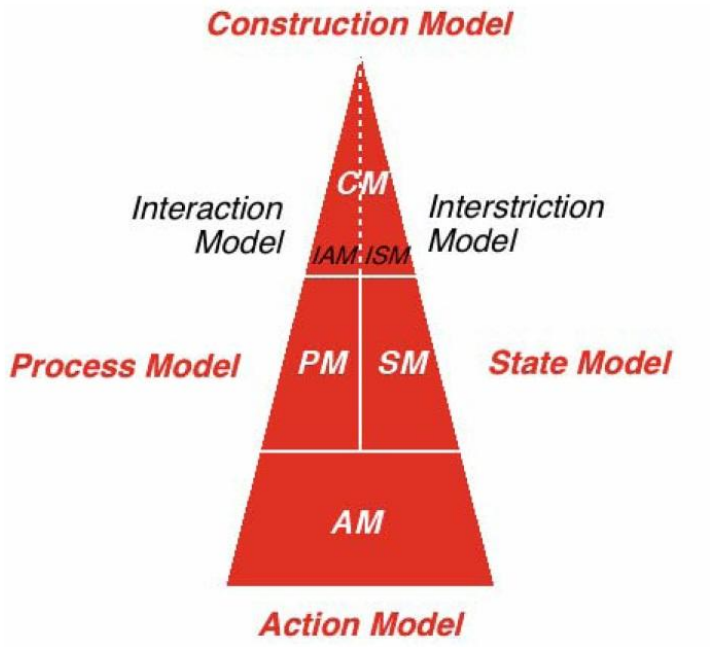


Image 12: Aspects Model [1]

- Construction Model; the construction model specifies the transaction types, their associated actor roles and information links between the actor roles and production and coordination banks. It's considered the most concise model and as such figures on the top of the pyramid. [1]

  The construction model can be also split in two, the interaction model were the active influences between actors are shown (execution of transactions), and the interstriction model were the passive influences between actors are shown (existing facts are taken into account). [1]

- Process Model; the process model contains, for every transaction present in the construction model, a specific transaction pattern and their causal and conditional relationships. [1]

- State Model; the state model specifies the state space of the production-world (the object classes, fact types, result types and ontological coexistence rules). [1]

- Action Model; the action model specifies the rules that serve as guidelines for the actors actions. It contains at least a rule for every agendum type (any step in the transaction diagram). [1]

The aspect models are produced anticlockwise, starting with the interaction model (IAM). [1]

"The first result for the enterprise ontology from any kind of documentation is a list of the identified transaction types and the participating actor roles, as well as the identification of the boundary of the enterprise.

From this knowledge, the IAM can be made. It is expressed in an Actor Transaction Diagram (ATD) and a Transaction Result Table (TRT)."

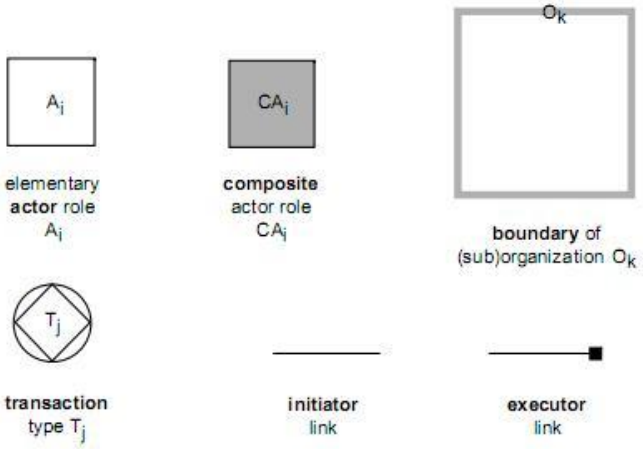


Image 13: ATD Basic Elements [3]

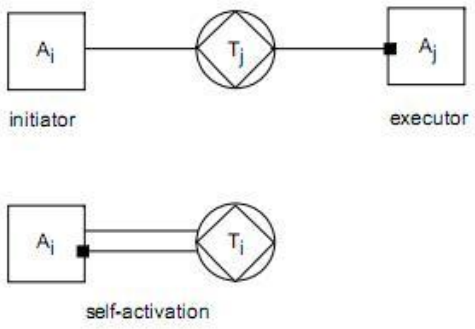


Image 14: ATD Basic Constructs [3]

"Next, the Process Structure Diagram (PSD) is produced, and after that the Action Rule Specifications (ARS)."

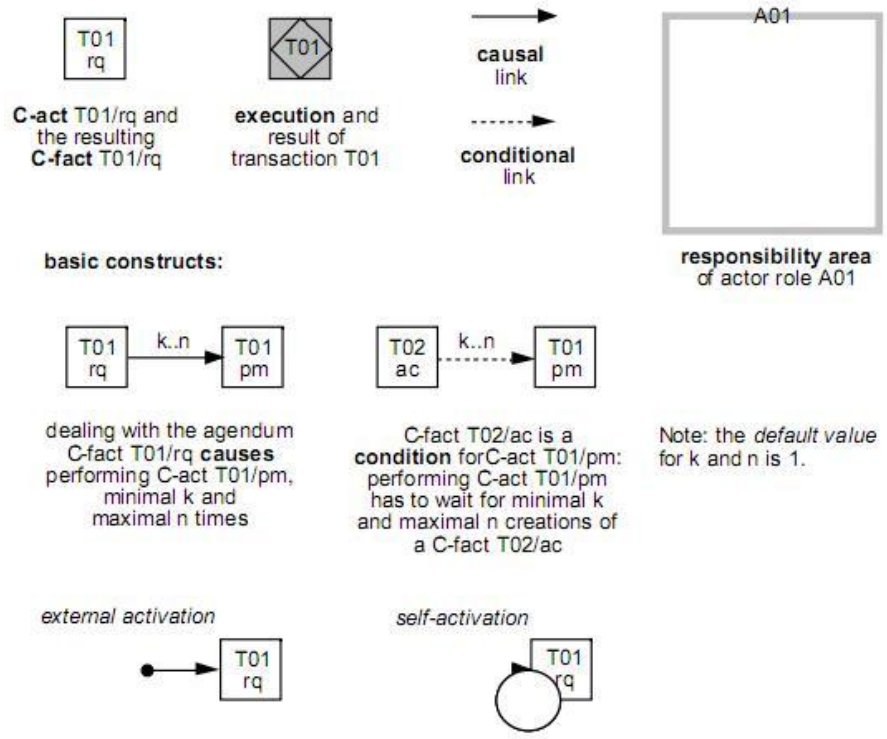


Image 15: PSD basic elements [3]

"The action rules are expressed in a pseudo-algorithmic language, by which an optimal balance is achieved between readability and preciseness.

Next, the SM is produced, expressed in an Object Fact Diagram (OFD) and an Object Property List (OPL)."

The OFD diagram uses the same notation and basic constructs as the WOSL that we have previously seen.

With the State Model produced *we are able to complete the PM with the Information Use Table (IUT).*

Lastly, the ISM is produced, consisting of an Actor Bank Diagram (ABD) and a Bank Contents Table (BCT).

Usually the Actor Bank Diagram is drawn as an extension of the Actor Transaction Diagram; together they constitute the Organization Construction Diagram (OCD)." [1]

The steps to acquire the needed information for aspects model are the following:

- *"The Performa-Informa-Forma Analysis (all available pieces of knowledge are divided in three sets, according to the distinction axiom)*
- *The Coordination-Actors-Production Analysis (the Performa items are divided into Coordination-acts/results, Production-acts/results, and actor roles, according to the operation axiom)*

- *The Transaction Pattern Synthesis (for every transaction type, the result type is correctly and precisely formulated; the Transaction Result Table can now be produced)*

- *The Result Structure Analysis (according to the composition axiom, every transaction type of which an actor in the environment is the initiator may be conceived as delivering and end result to the environment. Generally, the (internal) executor of this transaction type is initiator of one or more other transaction types, and so on)*

- *The Construction Synthesis (for every transaction type, the initiating actor role(s) and the executing actor role are identified, based on the transaction axiom; this is the first step in producing the Actor Transaction Diagram)*

- *6) The Organization Synthesis (a definite choice has to be made as to what part of the construction will be taken as the organization to be studied and what part will become its environment; the Actor Transaction Diagram can now be finalized)"* [1]

For our thesis purpose we will focus essentially in the three most important kinds of diagrams that are produced in the DEMO methodology; the Actor Transaction Diagram in the construction model the Process Step Diagram in the process model and the Object Fact Diagram in the state model.

2.1.6. DEMO Meta Model

In the organization self, the constituting organizational artifacts (constructs of an organization, like for example a business rule) are arranged in a manner that specifies all four aspects (construction, process, action and state). This arrangement between them has to obey a certain set of rules. The conceptualization of these rules is called the organization space. [7]

By organization space we understand a set of allowed organizational artifacts specified by the organization artifacts base (organization artifact kinds of witch instances, called organizational artifacts, can occur at a state base) and organization artifacts laws that determine the inclusion or exclusion of coexisting artifacts. [7]

This definition of organization space is similar to the definition of state space of an organization's production world that the WOSL specifies and we have previously seen. For this reason D. Aveiro in his thesis*"G.O.D. (Generation, Operationalization & Discontinuation) and Control (sub) organizations: a DEMO-based approach for continuous real-time management of organizational change caused by exceptions"* [7] found appropriate to use WOSL to define the organization space diagram designated as DEMO Meta Model.

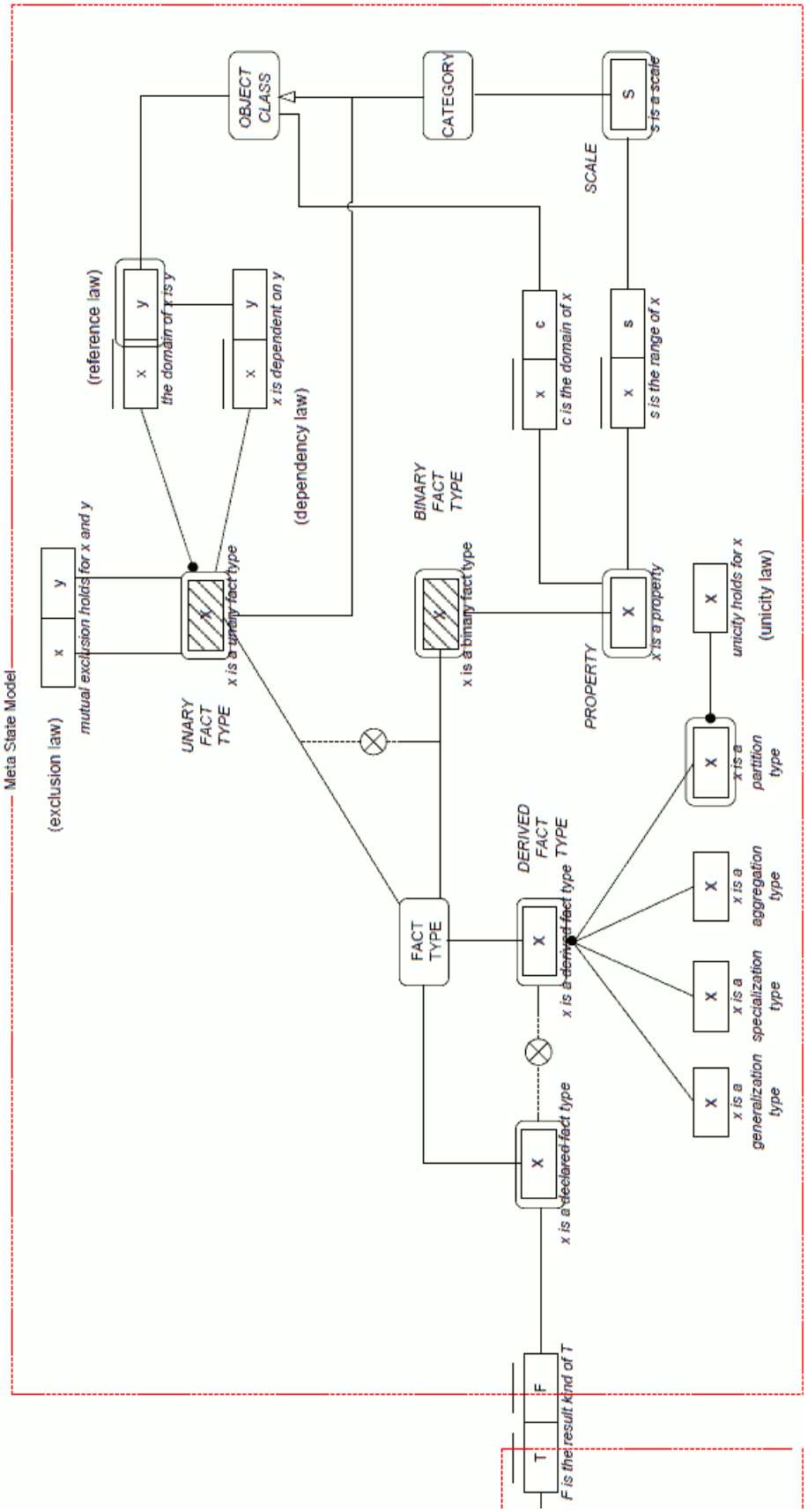


Image 16: Meta State Model [7]

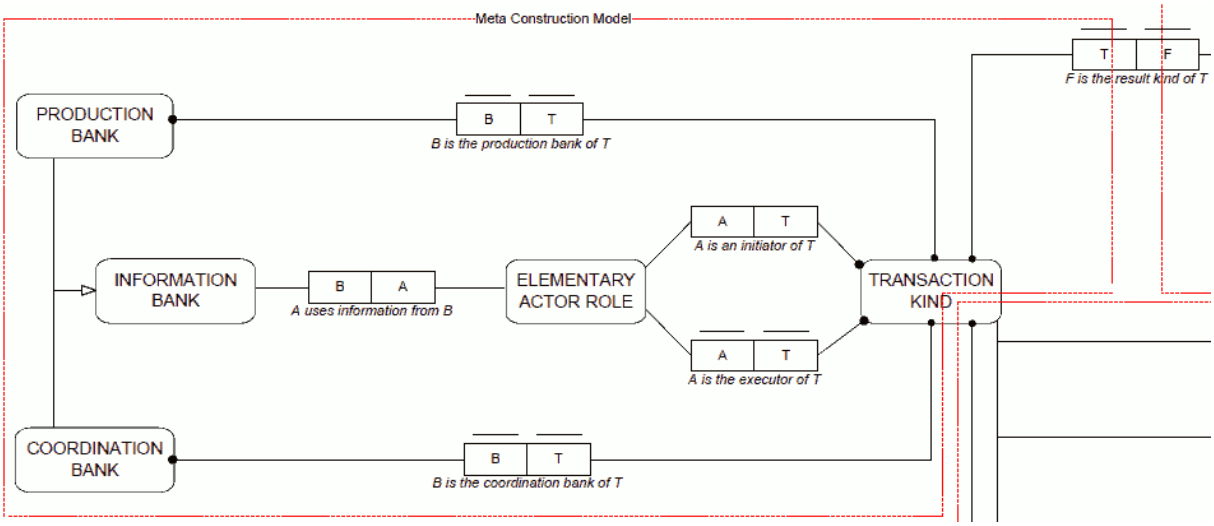


Image 17: Meta Construction Model [7]

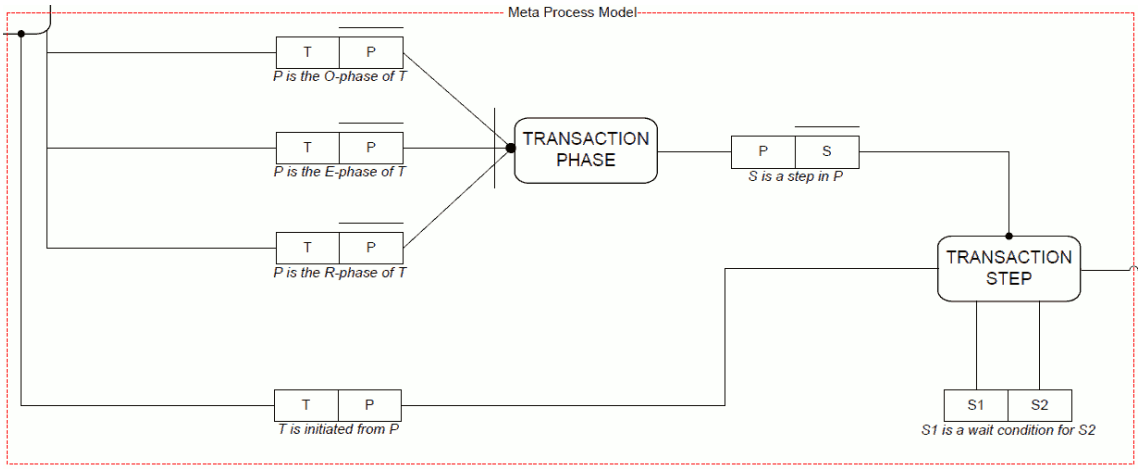


Image 18: Meta Process Model [7]

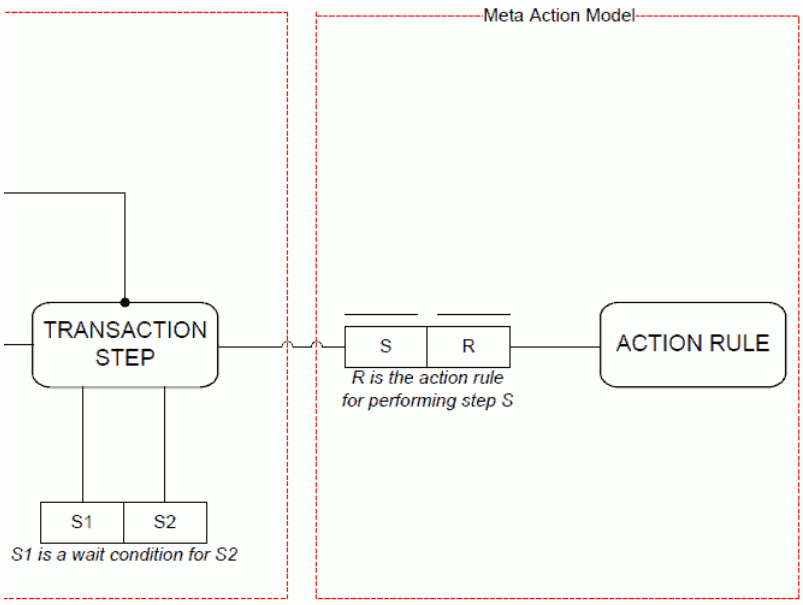


Image 19: Meta Action Model

2.1.7. Media Wiki

The MediaWiki, is a free web-based software written in PHP programming language developed by the Wikimedia Foundation and other parties. It is used in widely across the internet to power wiki websites including Wikipedia and Wikinews and uses a backend MySQL database. [16]

It is licensed under GNU GPL and is optimized to handle projects of any size, including the huge Wikipedia that holds over four million articles. To achieve such scalability multiple layers of caching are used database replication. [16][17]

"The software is also highly customizable, with more than 700 configuration settings and more than 1,800 extensions available for enabling various features to be added or changed. "[16]

In our implementation MediaWiki will serve as the base layer for the extension Semantic Mediawiki.

2.1.8. Semantic MediaWiki

To have the complete notion of what is Semantic MediaWiki and its purposes; we will quote its official website definitions;

"A semantic wiki is a wiki that has an underlying model of the knowledge described in its pages. Regular, or syntactic, wikis have structured text and untyped hyperlinks. Semantic wikis, on the other hand, provide the ability to capture or identify information about the data within pages, and the relationships between pages, in ways that can be queried or exported like a database." [18]

"Semantic MediaWiki (SMW) is an extension to MediaWiki that allows for annotating semantic data within wiki pages, thus turning a wiki that incorporates the extension into a semantic wiki." [19] "*While it appears to make things more complex, it can also greatly simplify the structure of the wiki, help users to find more information in less time, and improve the overall quality and consistency of the wiki. Here are some of the benefits of using SMW:*

- *Automatically-generated lists. Wikis tend to contain many aggregated lists; Wikipedia itself has thousands, like "List of metropolitan areas in Spain by population". Those lists are prone to errors, since they have to be updated manually. Furthermore, the number of potentially interesting lists is huge, and it is impossible to provide all of them in acceptable quality. In SMW, lists are generated automatically like this. They are always up-to-date and can easily be customized to obtain further information.*

- *Visual display of information. The various display formats defined by additional extensions, such as Semantic Result Formats and Semantic Maps, allow for displaying of information in calendars, timelines, graphs and maps, among others, providing a much richer view of the data than simple lists would.*

- *Improved data structure. MediaWiki wikis tend to make heavy use of categories for structuring data. While these are generally helpful, consider the category on Wikipedia called "1620s deaths"; if the information in these pages were stored using SMW, these categories could be replaced by simple semantic values, reducing the need for a complex classification system. In addition, if semantic markup within the wiki is stored within templates, otherwise known as semantic templates, a wiki can easily gain a solid data structure. And the Semantic Forms extension lets administrators create forms for adding and editing the data within semantic templates, thus making the addition of semantic information possibly even easier and more straightforward than regular wiki text.*

- *Searching information. Individual users can search for specific information by creating their own queries, supported via extensions like Halo and Semantic Drilldown.*

- *Inter-language consistency. In wikis that span multiple languages, like Wikipedia, there is often a great deal of data redundancy, which can lead to inconsistencies. For example, the population of Edinburgh at the time of this writing is different in the English, German, and French Wikipedias. If data is stored semantically, you could, for instance, ask for the population of Beijing that is given in the Chinese Wikipedia without knowing a single word of that language. This can be exploited to have different languages query one another's data, either for reuse or at least to detect inconsistencies.*

- *External reuse. Data, once it is created in an SMW wiki, does not have to remain within the wiki; it can easily be exported via formats like CSV, JSON and RDF. This enables an SMW wiki to serve as a data source for other applications, or, in the case of enterprise usages, to take over the role that a relational database would normally play. Through the use of the External Data extension and the result format Exhibit, one SMW-based wiki can even use the data from another, eliminating the need for redundancy between wikis. You can also query SMW's data via an RDF triplestore, using any of the available triplestore connector extensions. "[20]*

2.2. Design tools state of the art analysis

There are many modeling design tools out there to be used, but not all are fit for our purpose, so an analysis was needed to exclude those who didn't fit and point out those who fit better.

2.2.1. Xemod

Xemod is a software tool created for modeling the business process of organizations. This tool was created based on the DEMO methodology, therefore was a great candidate to be used in this thesis, so it was one of the first to be analyzed.

Xemod has an intuitive interface for someone with the basic knowledge of the DEMO methodology and has already built in rules to enforce the validity of the designs.

The issues with this software are the price, that is overwhelming when compared with the other analyzed options, and the limited options to export the files, (Microsoft Excel file, Microsoft Access file or image formats) that are hard to work with on the posterity to achieve what is needed.

2.2.2. Visio

Visio, the Microsoft modeling tool, worldwide known and used, with many potentialities including the ability to implement the DEMO diagrams by creating new sets of stencils.

Visio is intuitive and of easy use, allows the creation and update of new sets of diagrams and symbols in a relatively simple way, allows the creation of sets of rules for those diagrams and gives us many exporting formats such as the *.vdx (Visio xml).

On the downside Visio is not also one of the cheapest tools that you can find. Still, if willing to pay the amount asked for the software, it would be a perfect candidate.

2.2.3. Dia

Dia, an open source tool that can be used for all sorts of diagrams including all needed for the DEMO methodology, but with some limitations.

Although intuitive and of easy use Dia's interface lacks organization. Also it doesn't allow the importation of full diagrams, you have instead to create the symbols and them, in the program, create the diagram and import its respective symbols. Another problem lies in Dia's stability, the program crashes quite often.

The program does allow to export in *.vdx (Visio xml) but it doesn't save the information about how the shapes are connected which makes the work far harder since it would force the use of coordinates to find out.

So despite being free and allowing good exportation formats to work with like *.vdx, Dia does not meet the requirements to develop this thesis properly.

2.2.4. Kivio

Kivio is another open source tool that can be used in many business areas including implementing the DEMO methodology diagrams that isn´t however available to windows.

Kivio is also intuitive and of easy use and has a good interface. To create diagrams and shapes there is specific software (that is not open source) that can be used, but there is also the possibility to code the shapes using a simple text editor.

In Kivio we also have many formats that we can export the diagrams to, including the xml but unlike Dia, in Kivio's xml there is all the information needed to know the connections present in the diagram.

Kivio doesn't however allow the implementation of rules regarding the shapes or diagrams.

In the end, even not being the software with most potentialities, Kivio is still a good candidate due to the fact of being open source.

2.2.5. Others

There were other designing tools tested but some didn't come close to meet the needed requirements to implement the DEMO diagrams such as Open Office Draw, StarUML and ArgoUML. Although very interesting free designing tools their potential for new diagram designs options isn't one of their strengths.

Still there were others, not mentioned before, that were fit to implement the DEMO diagram. From a fusion of those with the ones mentioned before (the ones most likely to fit the needed profile) resulted the following table where we evaluate the points we consider most important to meet our goal:

Table 1- Design tools analysis

Program	License type	Interface	Able to create new diagram types	Able to create new symbols	Able to implement DEMO diagrams	Able to establish rules for the created elements	Supported exporting types
Visio 2010	Commercial($249.99) [8]	Intuitive and of easy use	Yes (relatively simple)	Yes (relatively simple)	Yes	Yes	Many including *.vdx (Visio xml)
Xemod	Commercial (€ 695,00 ($926)) [9]	Intuitive and of easy use	No	No	Yes	No (but has them implemented already for DEMO diagrams)	Only as Excel tables, Access entities or image.
Smartdraw 2010	Commercial($197) [10]	Intuitive and of easy use	Yes (relatively simple)	Yes (require some work)	Yes	No	Only as image
Dia v.0.96	Open Source (GPL) [11]	Intuitive and of easy use but with lack of organization	Yes (relatively simple)	Yes (simple but very limited)	Yes (limited)	No	Many including *.vdx (Visio xml)
Visual-paradigm	Commercial($358.5) [12]	Intuitive and of easy use	Yes (relatively simple)	Yes (but complex)	Yes	Yes	Many including *.xml
Enterprise Architect version 7.5	Commercial($199) [13]	Complex interface that requires learning first	Yes (but very complex)	Yes (but complex)	Yes	Yes	Many including: C, C++ , C#, Java, Python, System C, Visual Basic, PHP
Kivio	Open Source (GPL)[14]	Intuitive and of easy use	Yes (relatively simple)	Yes (relatively simple)	Yes	No	Many including *.xml

3. Implementation

3.1. Example Case Used

To illustrate the implementation of our prototype we a library scenario taken from *Enterprise Ontology* - Dietz, 2006[1] will be used. Our main focus will be on the Actor Transaction Diagram (ATD) and the Object Fact Diagram (OFD) represented below as these are the two main diagrams in the DEMO Methodology. The solution diagrams presented in *Enterprise Ontology* - Dietz, 2006[1] were remade using a more recent notation available in *Way of Working* - Dietz, 2009[5].

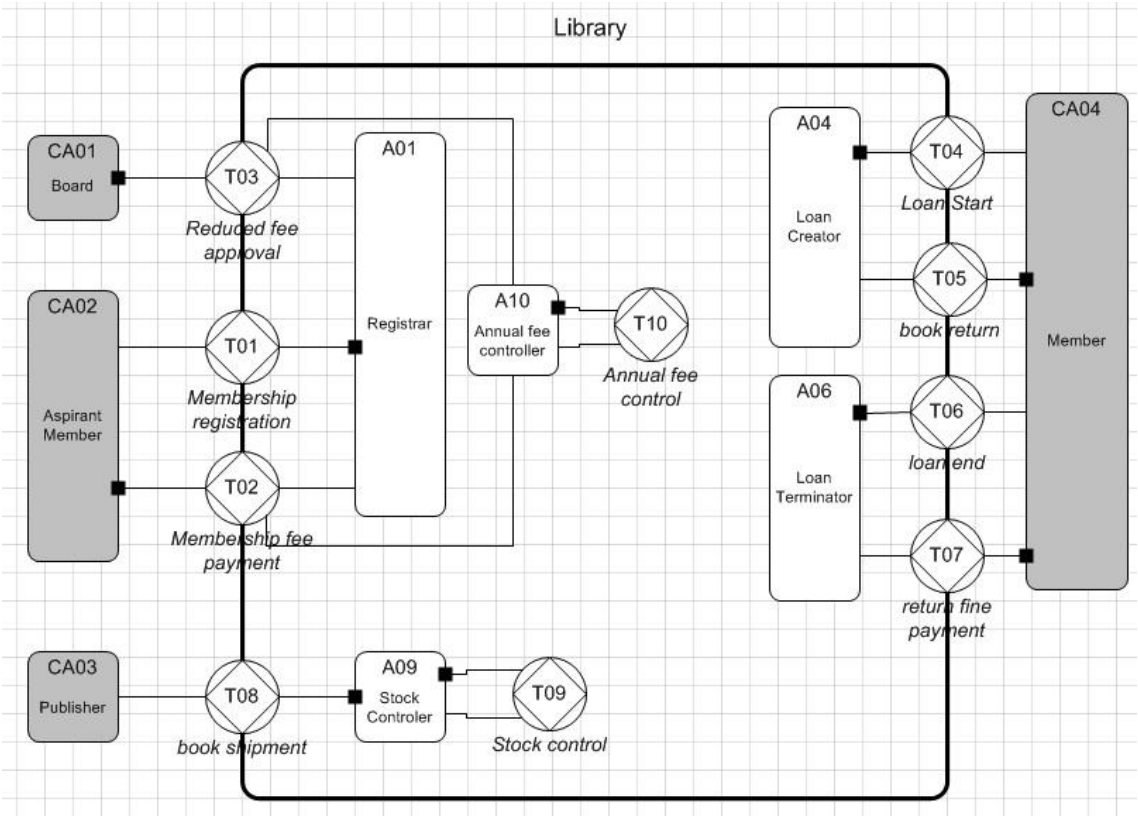


Image 20: Library ATD Diagram (Visio)

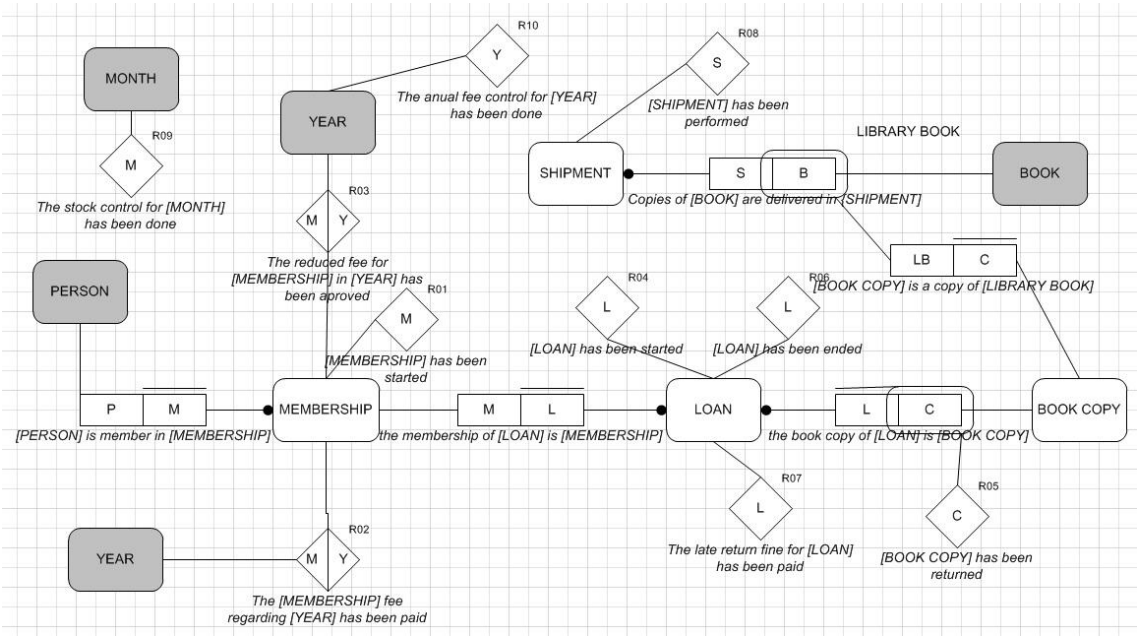


Image 21: Library OFD Diagram (Visio)

3.2. Validation rules for diagrams

One other thing that was analyzed was the possibility to apply specific rules to the diagrams, such as an internal actor can only execute one transaction. Those kinds of rules are already "built in" in Xemod, and are doable in Visio 2010; however in the open source alternatives (Kivio and Dia) it is not possible to implement them, therefore the idea was set on hold.

3.3. Changes in the DEMO Meta model

After a careful analysis to the DEMO Meta model it was noticed that there were some aspects needed for modeling that were missing, therefore it was proposed some changes to the original model. Those changes were, in the Meta Construction Model the differentiation between elementary and composite actor roles, production banks, coordination banks and in Meta State Model the distinctions between object class, derived object class and external object class. The resulting Meta model diagram was the following:

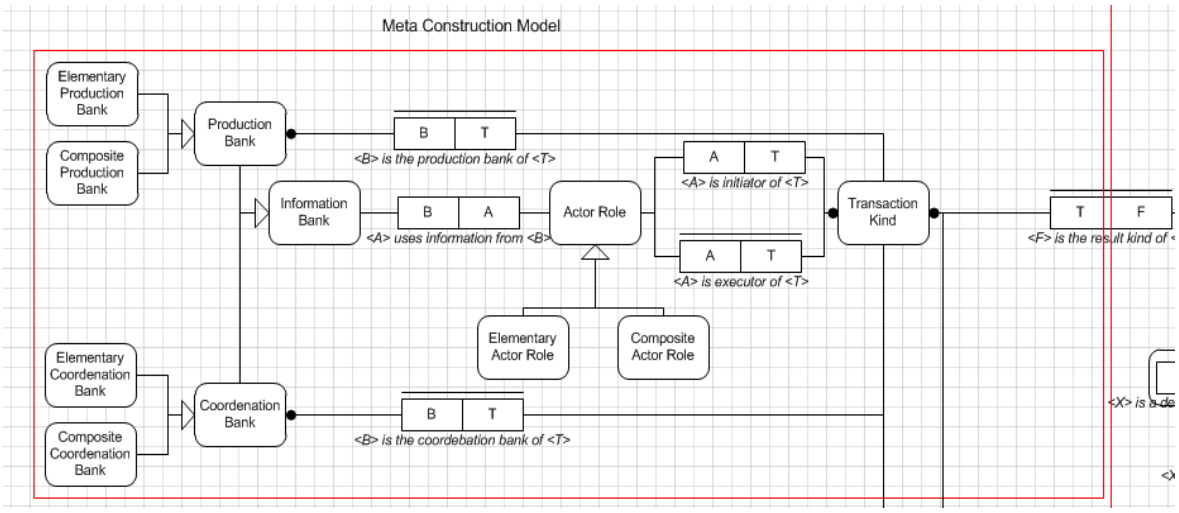


Image 22: Proposition for the DEMO Meta Construction Model

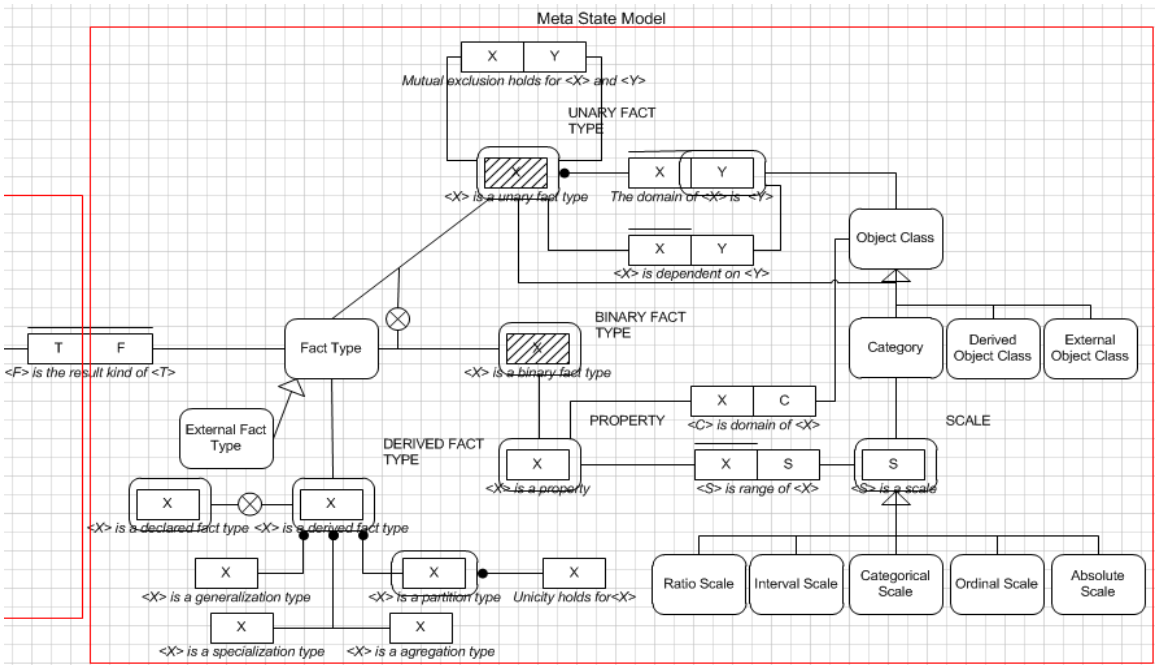


Image 23: Proposition for the DEMO Meta State Model

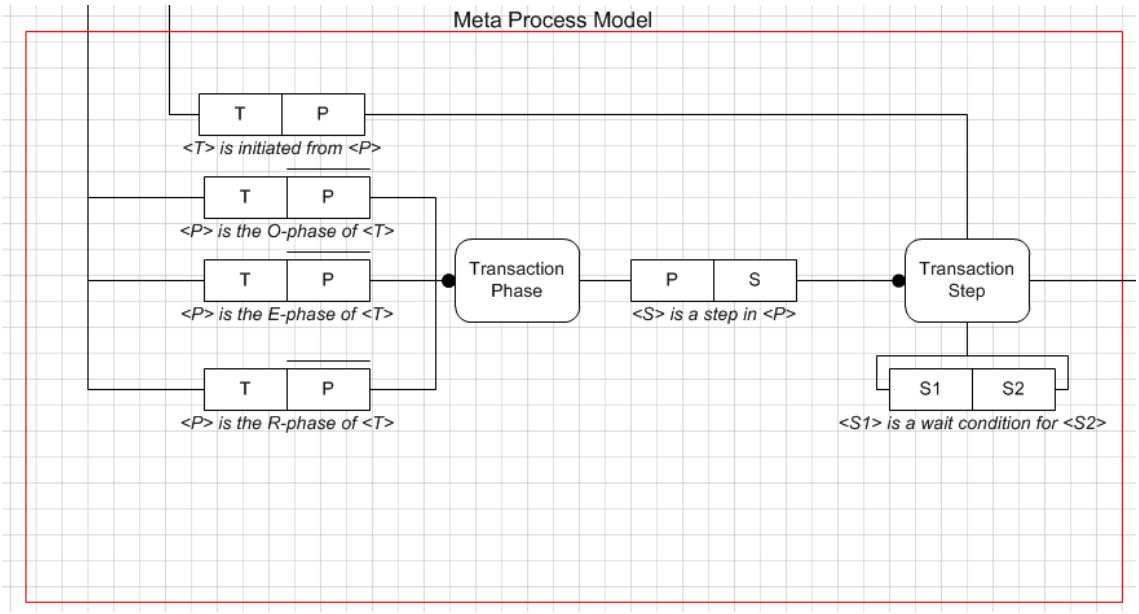


Image 24: DEMO Meta Process Model

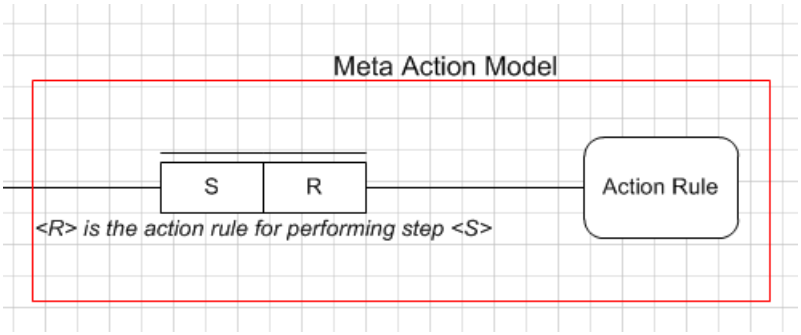


Image 25: DEMO Meta Action Model

3.4. XML implementation of ATD and OFD diagrams

It was theme of a Master's thesis in computer science by Yan Wang the "*transformation of DEMO models into exchangeable format*"[21]; here we gather those ideas of how to transform the DEMO methodology into XML and try to put them in a practical creation (with the respective changes to accommodate the Meta model differences). In that thesis it's studied how to transform without loss each of the four models that compose the DEMO into a XML document. This is a valuable contribution to this thesis because it gives a starting point about how to transform the XML received from the designing tool into something that we know to have all the needed data to recreate that diagram somewhere else, in this case Media Wiki.

3.5. Stencils Implementation

When testing new tools for a possible implementation of DEMO diagrams there was the need to try to recreate the shapes we were going to use on them. A complete stencil set was already available for Visio and for Xemod but for all the others such thing didn't exist. Being Visio and the open source tools (Dia and Kivio) the three tools more likely to be explored, the first for offering the most potentiality, the second for being the free tool with most potential for Windows users and the third for being the free tool with most potential but limited to Linux, there was the need to replicate the stencil sets for the open source tools in order to fully explore their potential.

3.6. Xemod Implementation

Even though Xemod was not a feasible tool for our objectives we still decided to fully implement the library example in order to have a clearer view of what could be done in a software designed to work solely with DEMO and perhaps give us some insight on ways to improve our stencils before their creation in the open source tools.

Unlike all the others design tools we tried, Xemod was specifically design for DEMO, and as such it was not only limited to diagrams, but instead allowed to implement the whole process.

Now even this software had its setbacks, in the Transaction Result Table, after having creating a result, his cardinality could not be changed, making it impossible to convert a binary result into a unary result and vice-versa. Other than this, while designing the PSD diagram, if a connecting mistake was made, the program just stopped responding instead of giving an error message.

After the whole process of implementing our Library example in Xemod, we exported it to access database format. Then we could do a detailed analysis to every table that was necessary to indentify each process, especially those related with the diagram steps, and how they related the facts.

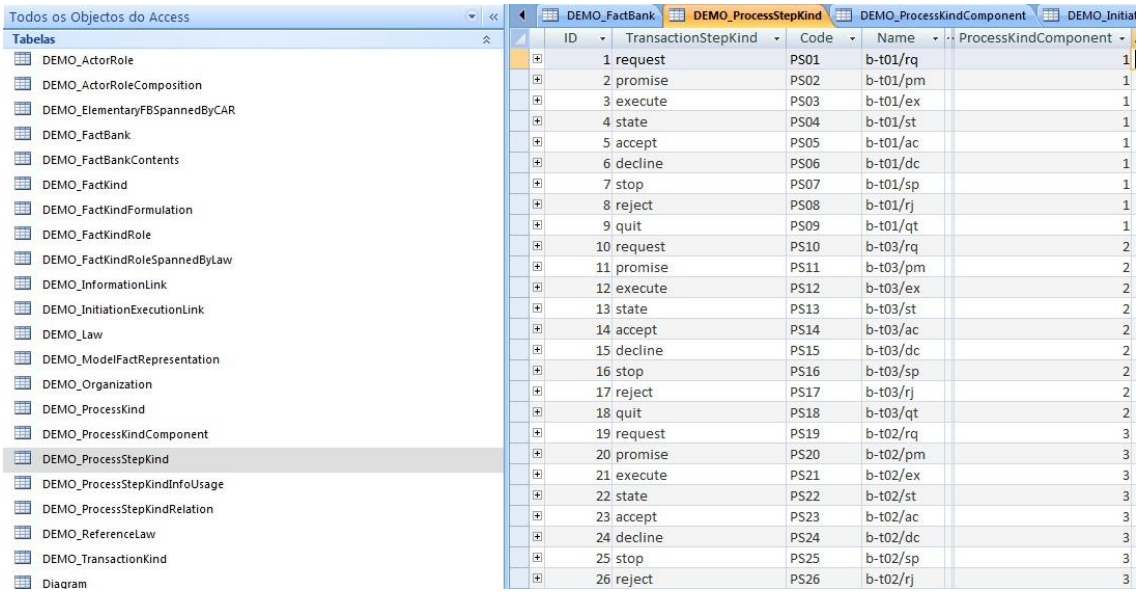


Image 26: Access database of Library exported from Xemod

But as we previously though, in neither the exportable formats it was possible to recreate any sort of diagram, as we only had the content of the shapes and their connections. We lacked the information about their positions, sizes or connection places.

3.7. DIA Implementation

In DIA, the software offers the option to create its own shapes in it and latter use them, and so it was done for the ATD. For our basic stencil we created the shapes of an Actor, Composite Actor, Transaction and Boundary.

In this creation process, the first issue we noticed was the text field's limitation. DIA was limited to one text field in each shape, so in order to make the ATD stencil possible

we had to adapt the diagram shapes to a single text field. This limitation would however be very problematic if we decided to implement the OFD diagram.

The second problem encountered in the shapes creation was the fact that unless you did a mouse-over, you had no labeling on the shapes, making for example a boundary and an actor virtually undistinguished by sight. But still, this was not a major flaw, and work could proceed.

The third problem was the fact that it was impossible to create connectors, so the connectors used in our diagram would have to be the default connector changing then, if existed for that case, the tip to the desired shape. This would mean that the diagram designer would have to know the notation for each connector in order to make a complying diagram.

With all four starting shapes created, we needed to create a stencil. This was an easy process, we just had to go to the "sheets and objects" option in the "File" tab, and create a new stencil, were we gave it a name and a small description.

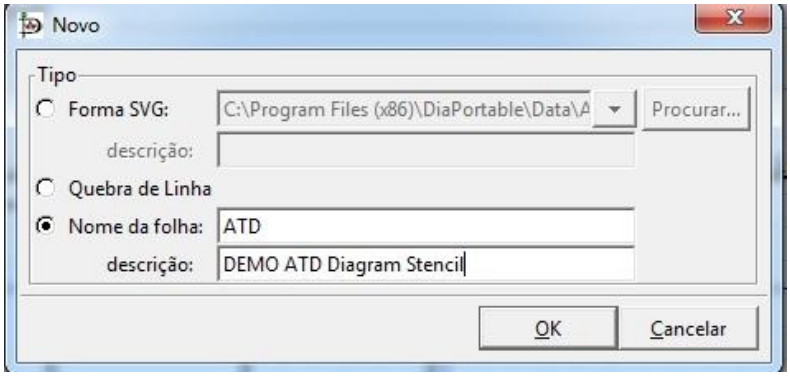


Image 27: DIA Stencil Creation

Now, to add the shapes, the process was exactly the same, but the selected option would be the first, "Shape SVG" and then search for the shape file previously created. DIA does not allow multiple importations at once, so this step has to be done for every shape you intend to use in each stencil.

When we started to create our diagram, two new problems were found, the first was the fact that, when the text exceeded his preset delimitation, the shapes would grow out of proportion, reaching huge sizes when compared to everything else, and the second was that to apply the boundary we needed to work with layers, as although shapes with no filling are possible, to select the ones in the back, they can't be under anything, nor you can move the shapes (to front or back) after they had been set.

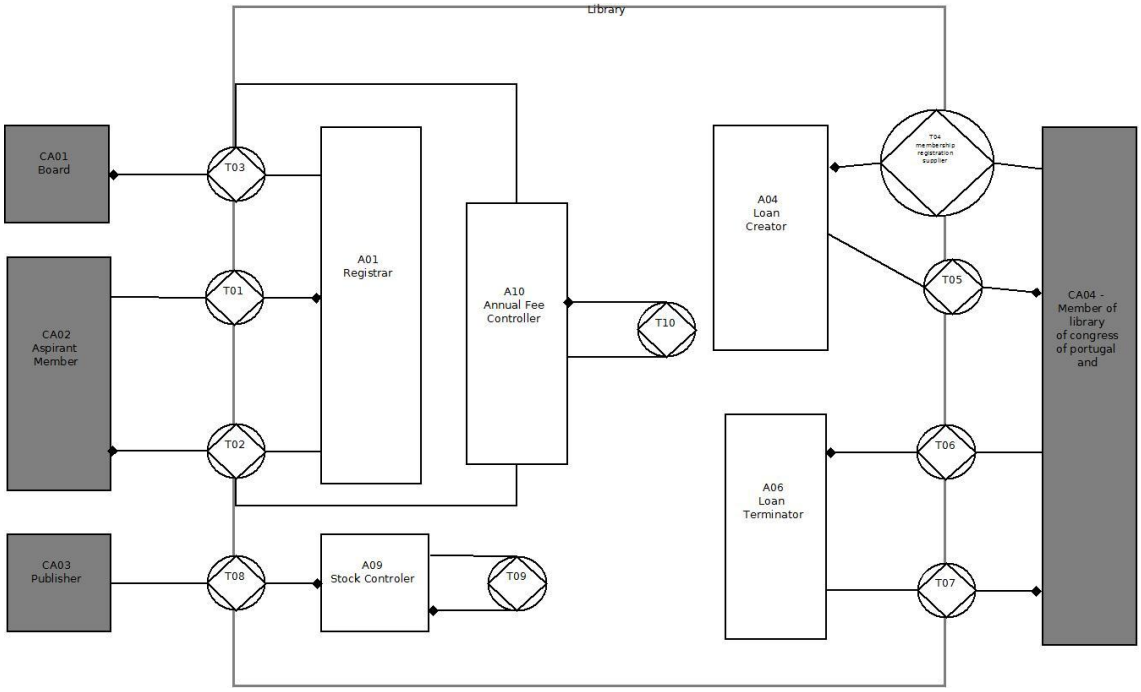


Image 28: DIA Implementation of Library ATD

More problems were found during and after the implementation. The program crashed often when placing shapes on the drawing board, and if by any reason you tried to open this diagram in another computer with DIA installed but without the stencil set in place, the shapes would not appear.

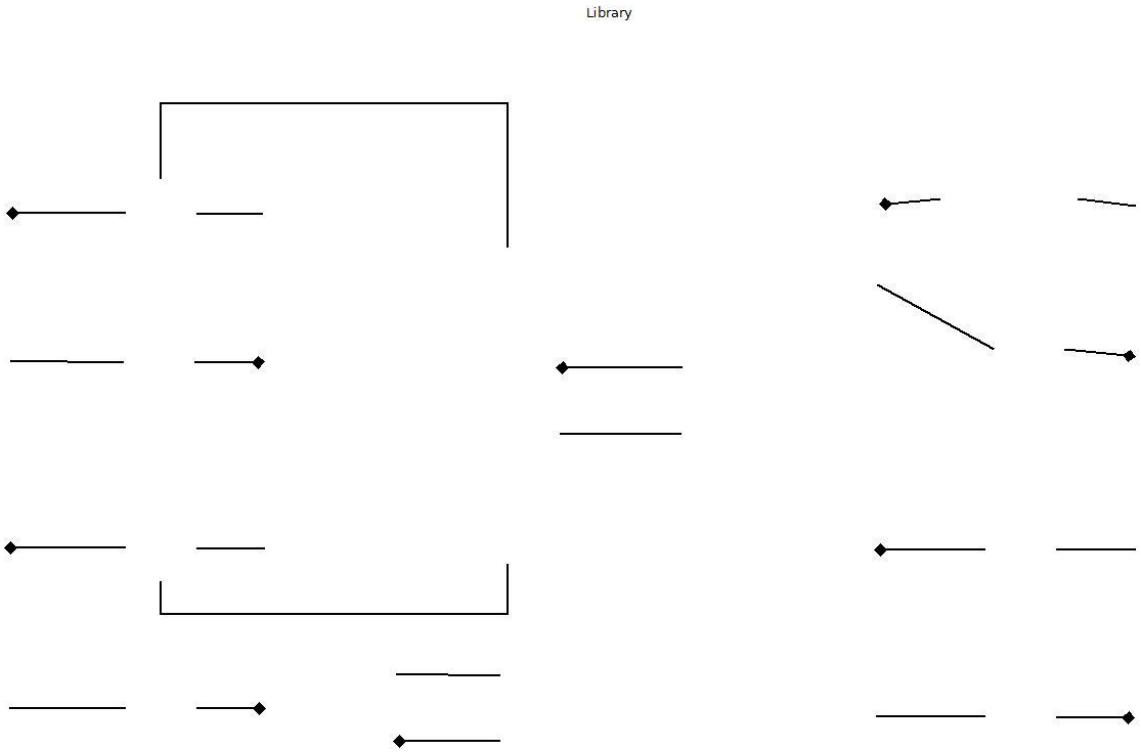


Image 29: Importation of Library ATD in DIA with no stencil set

However DIA had a major issue that led to its abandon, the fact that we couldn't relate the connected shapes, making it impossible to use unless relaying fully on coordinates.

With this, the work was focused on Kivio.

3.8. Kivio Implementation

Unlike Dia, in Kivio, the new shapes cannot be created in the software itself; instead, there is paid software for that purpose. But there is a second solution, which is to code the shapes with no graphical help simply by using Kivio's xml language in a notepad. This second solution was the one followed.

The first step here was to create a new folder for our diagrams methodology in the shapes folder of Kivio, in this case, one named "DEMO". In there we could create the subfolders for each of the diagrams such as one for ATD and one for OFD, but in the same folder we have a need for a "desc" (description) file that contains the info about our set of diagrams and as an option we can add an image to represent that very same main set.

This "desc" file is written in XML using Kivio's tag sets and contained the following simple information:

```
<?xml version="1.0"?>
<KivioStencilSpawnerSetCollection>
        <Title data="DEMO Diagrams"/>
        <Id data="Kivio - DEMO Diagrams"/>
</KivioStencilSpawnerSetCollection>
```

Now, inside each diagram folder (such as ATD) we have tree kinds of files:

1. The "desc" file, simple as the first one, in this case containing the following:

```
<?xml version="1.0"?>
<KivioStencilSpawnerSet>
        <Title data="Actor Transaction Diagram Shapes"/>
        <Id data="Duarte Pinto - Actor Transaction Diagram Shapes I"/>
         <Description>the basic stencils needed to create Actor Transaction Diagrams
        </Description>
</KivioStencilSpawnerSet>
```

2. An optional image *.xpm file for each shape.

3. And a *.sml file for each shape containing its data. Kivio's XLM recognizes some types of basic shapes such as rectangle, circle or square, and, the ones that are not recognized, we can draw using lines. As an example of this code we have the Composite Actor shape:

```
<?xml version="1.0"?>
<KivioShapeStencil creator="Duarte">
	<KivioSMLStencilSpawnerInfo>
		<Author data="Duarte Pinto"/>
		<Title data="Composite Actor"/>
		<Id data="CompositeActor"/>
		<Description data="ATD Composite Actor"/>
		<Version data="0.1"/>
		<Web data="www.koffice.org"/>
		<Email data="dnprules@gmail.com"/>
		<Copyright data="Copyright (C) 2010 Duarte Pinto. All rights reserved."/>
		<AutoUpdate data="off"/>
	</KivioSMLStencilSpawnerInfo>
	<Dimensions w="72.0" h="54.0"/>
	<KivioConnectorTarget x="0.0" y="0.0"/>
	<KivioConnectorTarget x="36.0" y="0.0"/>
	<KivioConnectorTarget x="72.0" y="0.0"/>
	<KivioConnectorTarget x="72.0" y="27.0"/>
	<KivioConnectorTarget x="72.0" y="54.0"/>
	<KivioConnectorTarget x="36.0" y="54.0"/>
	<KivioConnectorTarget x="0.0" y="54.0"/>
	<KivioConnectorTarget x="0.0" y="27.0"/>
	<KivioShape type="Rectangle" name="CompositeActor" x="0.0" y="0.0" w="72.0" h="54.0">
	<KivioFillStyle colorStyle="1" color="#736F6E"/>
	</KivioShape>
	<KivioShape type="TextBox" name="Text"   x="0.0" y="0.0" w="72.0" h="54.0"/>
</KivioShapeStencil>
```

By manipulating this code, it was possible to create the stencil sets for ATD and the OFD that will be then used for further develop of this thesis.

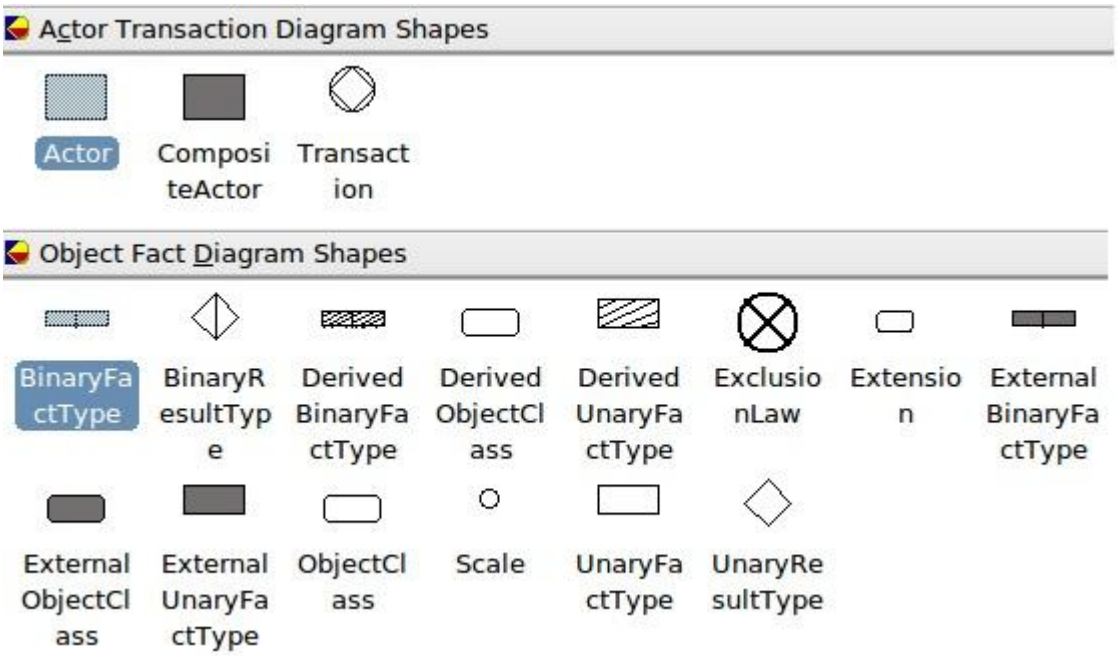


Image 30: Kivio Stencils

After having the stencils, the ATD and OFD diagram implementation in Kivio was straight forward.

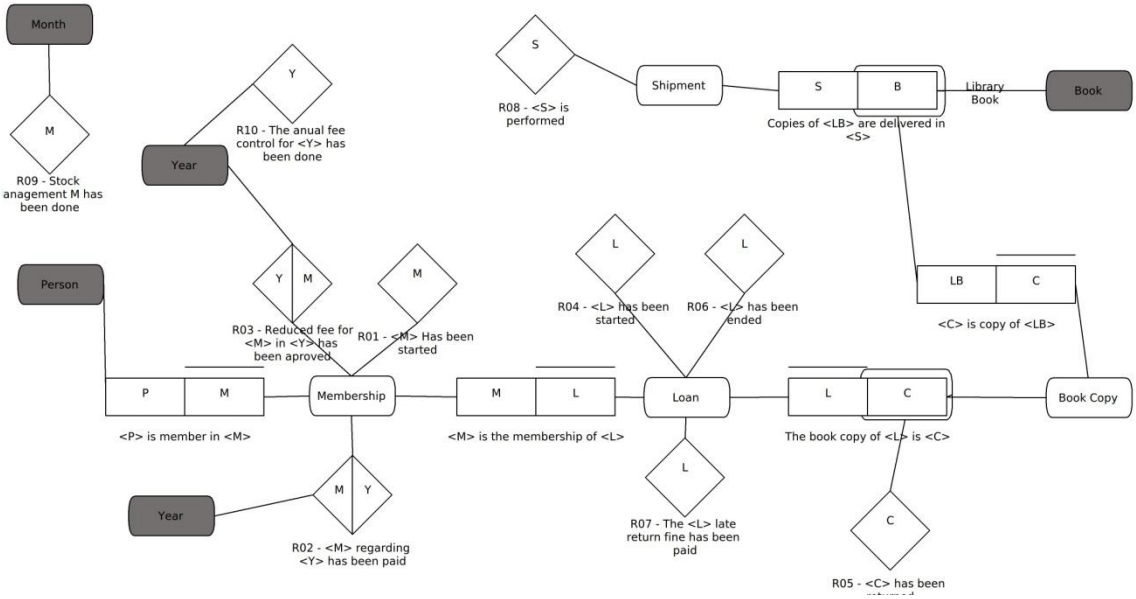


Image 31: Library representation in Kivio

In the OFD there were some minor changes to the notations like the result type number being moved to under the shape as placing the text on the top right corner of the shape was not possible due to shape size constrains.

3.8.1. Kivio PHP parser

The PHP parser is the practical implementation of the previous point, the first objective was to get the raw XML from the design tool and transform it into a format that can be understood, in this case, the one proposed by Yan Wang on Transformation of DEMO models into exchangeable format thesis. Later, this new formatted XML would be used to automatically generate Semantic Media Wiki pages that represent the whole information contained in the diagrams but in an easy to understand/update format that any member of the organization can use.

The PHP code of this parser can be found in full size on the appendixes of this thesis.

3.8.2. Kivio problems

Although a viable solution Kivio also had its issues, the first being the text fields limitation. After being set the size in the stencil, if the text field by any reason is insufficient to the text itself makes part of it not to become visible. This is a major issue in diagram interpretation.

Another issue was the connectors, although you could create shapes, the connectors were limited to the existing ones, having to adapt the connector ends to your needs. If true that all representations were still possible, the lack of the connectors in the stencils made it not only hard for an unfamiliar user to get the right notation, but also very time consuming.

Kivio development by KOffice had also been placed on hold, making his compatibility with future OS and formats another problem to be considered.

Finally there was the clutter in the exported file, although not as much as a exported *vdx file from Visio, it was still a great amount that had to be filtered in every shape and connector in order to acquire only the valuable info for our task at hand.

For these reasons we decided to place Kivio on hold and try Visio to see if it was a better candidate. This does not mean Kivio was not still a possible free solution to the problem, but perhaps was not the better equipped tool to allow us a fast development with a great margin for progression.

3.9. Visio Implementation

In Visio we had part of the work already done, as stencils for DEMO diagrams modeling were already available. The other plus side of working with Visio is that it was not a first impression with the tool; it had been used many times throughout the years by us including DEMO modeling itself. Still as every other tool in this case there were issues to solve

The first problem to solve, after the implementation of example diagrams of each type we were working at (ATD and OFD), was how to access the information relative to each shape.

Visio, as mentioned before, has already and export preset for xml, the *.vdx format, but there is a huge problem with this, there is absolutely no formatting on the output, and the amount of unneeded clutter is huge. Just to leave a general idea, to an export from our OFD model we obtained 1627 XML lines, reaching some of these lines some astonishing 741318 characters.

Just like it had been done in Kivio the first step here was to filter the variables we actually needed from the files, for that a PHP parser was created that striped the file from most of its clutter. After a while this task were starting to be overwhelming, so the search for an easier solution began.

The solution found was to use (VBA) Visual Basic for Applications. VBA is an implementation of Microsoft's Visual Basic that is available in the Microsoft Office Programs and also in Visio. In each application you can use it by creating a macro and it allows you to access in the case of Visio, pretty much everything that is on your diagram, may it be property's, coordinates, linked shapes, connector points, and so on, the list continues for a long, long amount of properties.

All this was also available in the *.vdx file, but now we could gain it easy access and filter just the ones we needed for our task in hand.

At this point one of the main questions arisen, what did we exactly need so we had enough information to create the Semantic Media Wiki pages for each shape, but also, to then recreate this exact same diagram elsewhere.

To accomplish such task the way encountered was to create a "Universal Visio Meta Model for DEMO" that would fit any possible diagram in DEMO and also new diagrams, shapes and connections, that not existing yet maybe be a solution to future developments.

The need for it to be abstract enough to include everything but still concrete enough to set boundaries was clear, so we did our best to accomplish it.

3.9.1. PSD Stencil

Unlike the ATD and OFD, the PSD stencil available for Visio was out of date, and did not comply with the new notation used in (Way of Working - Dietz, 2009).

For that reason we decided to build our own to comply with the PSD representations available in such document. The following image shows all the shapes that were created in the new stencil in so that it considered all the possible needs in the new notation.

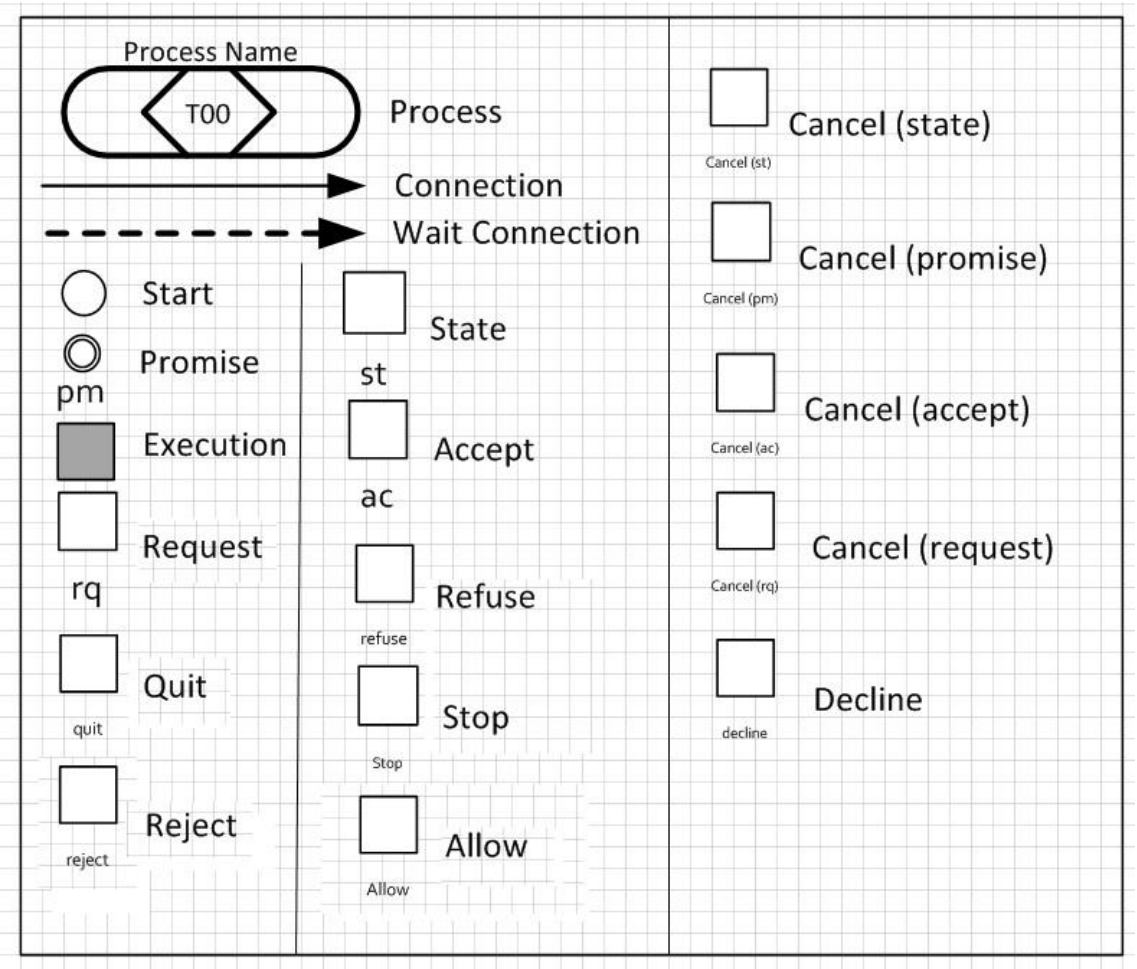


Image 32: New PSD stencil - shapes list

3.9.2. Universal Visio Meta Model for DEMO

The main purpose of this Universal Visio Metal Model for DEMO (UVMMD) was to serve as to guideline to what information was really relevant in a diagram and provide the necessary base in case any new diagrams were added to the DEMO methodology.

This model was specified using the World Ontology Specification Language (WOSL) and it will be presented in segments due to the obvious size constrains that we have to follow. To help acknowledge those segments the full diagram is presented bellow with the parts numbered with the other they will be explained.

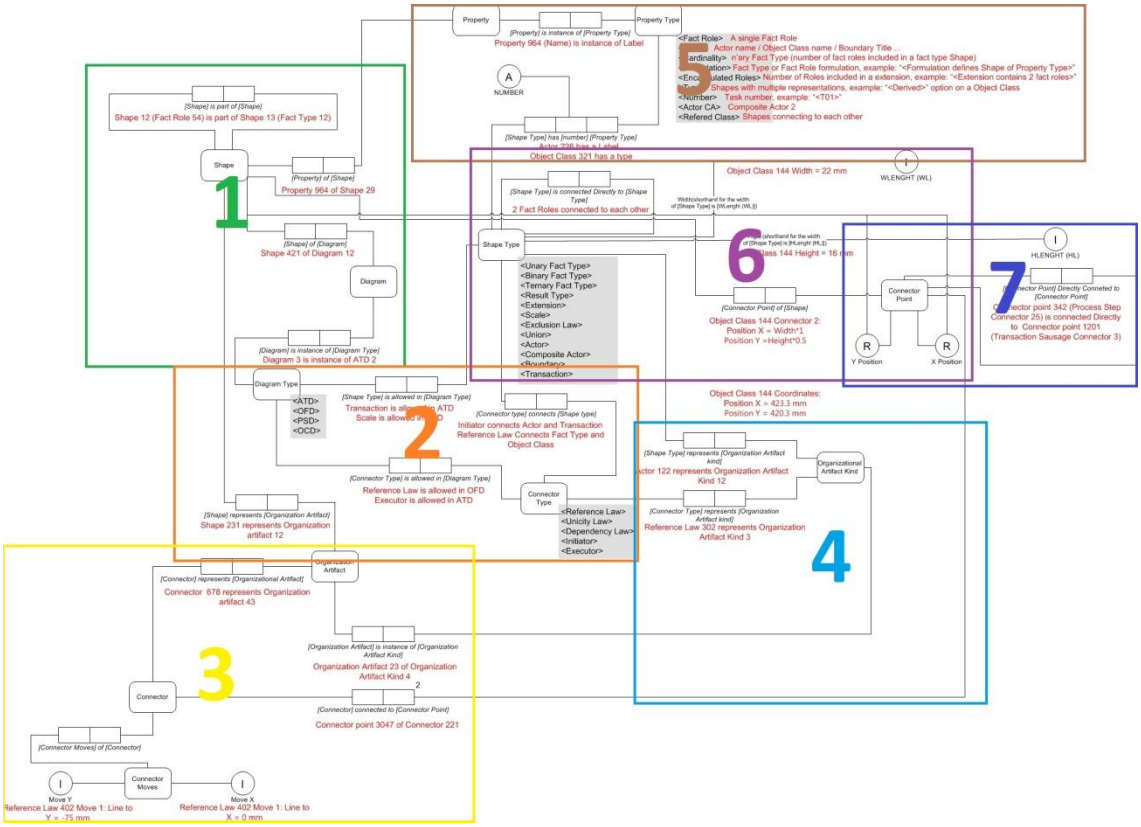


Image 33: Universal Visio Meta Model for DEMO

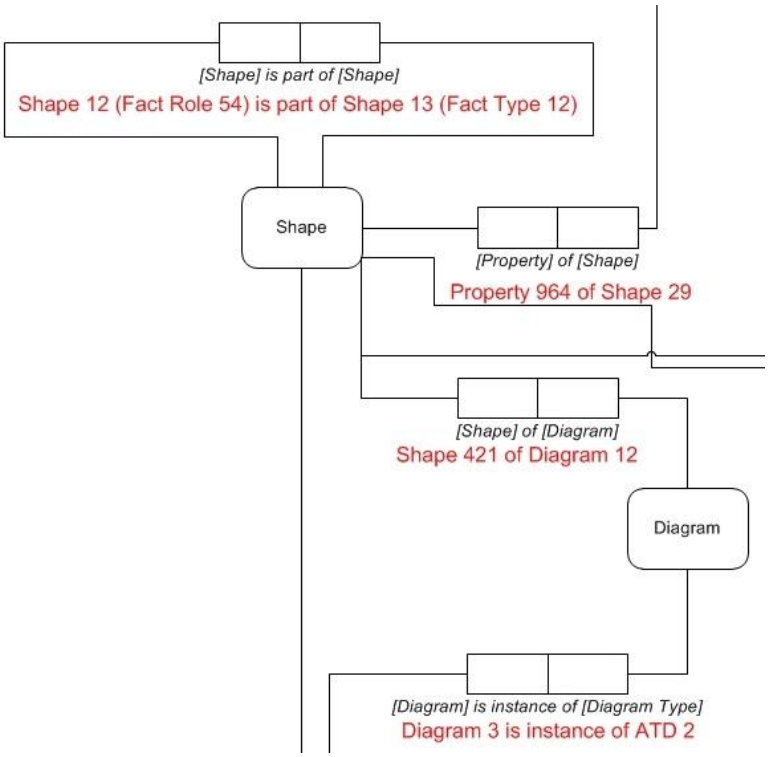


Image 34: Universal Visio Meta Model for DEMO – Part 1

Object class "Shape" – All diagrams are composed by shapes, they are the key of representation. This object class "Shape" is a concrete representation of the abstract concept object class "Shape Type" that we will see further ahead. An example of this could be in an ATD diagram the "Composite Actor 8" of the shape type Composite Actor. For us "Composite Actor 8" tells very little as we confine on shape properties like name to identify them but this is the way shapes are identified in the inner works of Visio.

Binary fact type *[Shape] is part of [Shape]* - A shape can be part of another shape, making a greater agglomerate of shapes, but still itself a shape. As an example of this we have the OFD Fact Type that is composed by one or multiple fact roles like the fact type *[PERSON] is member in [MEMBERSHIP]* composed by the fact roles [PERSON] and [MEMBERSHIP].

Binary fact type *[Property] of [Shape]* – Every Shape has properties; these properties may vary between shapes, but can also be shared by multiple shapes. Using our library OFD diagram as an example of this binary fact type, we can think of "*[PERSON] is member in [MEMBERSHIP]"* a property of the property type formulation that belongs to the shape "Binary fact type 193" (being "Binary fact type 193" the shape of the shape type Binary fact type represented in that diagram).

Binary fact type *[Shape] of [Diagram]* – Shapes are part of a greater picture, in our case diagrams. Every shape needs a context for its existence to make sense. As an example of this binary fact type we have the object class "PERSON" of diagram "Library OFD".

Object class “Diagram” – The Object Class “Diagram” is the concrete representation of the abstract concept object class “Diagram Type”. An example of such representation would be “Library OFD” or even “Universal Visio Meta Model for Demo”.

Binary fact type *[Diagram] is an instance of [Diagram type]* – When modeling a diagram, we don’t usually randomly pick a set of shapes and connectors to use, in most cases there are preset “templates”, and this is what is being represented in this relation. As an example this “Library OFD” or even “Universal Visio Meta Model for Demo” are instances of the diagram type OFD.

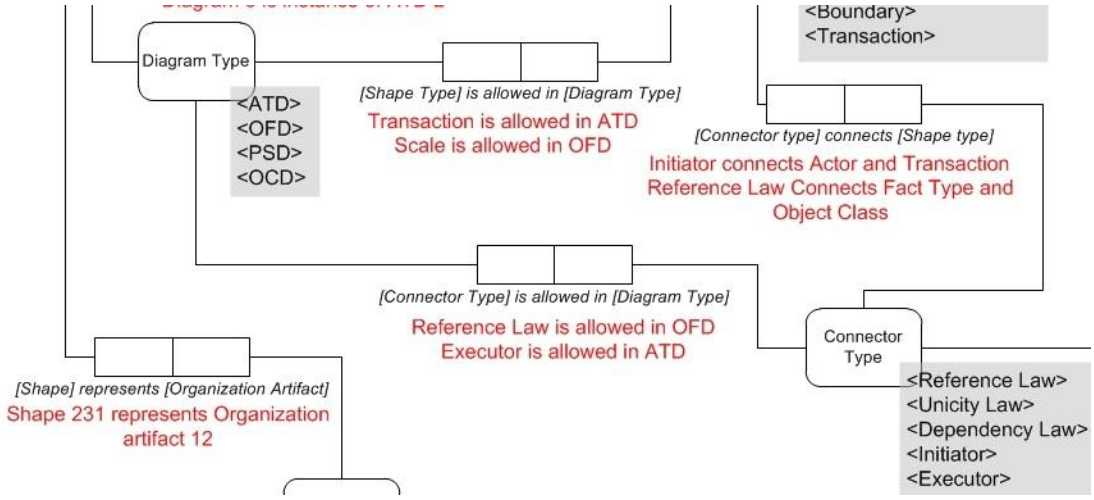


Image 35: Universal Visio Meta Model for DEMO – Part 2

Binary fact type *[Shape] represents [Organization Artifact]* – In this Binary fact type we enter the DEMO context of our modeling. An organization artifact is a “rule of application” for a shape or a connector. For this Binary fact type an example could be in an ATD diagram actor 183 (a shape of the shape type actor with the label “Board”) represents actor 183 initiates 12 transaction 97 (a shape of the shape type transaction with the label “Reduced Fee Approval”)

Object class “Diagram Type” - Diagram type is the “template” behind the diagram, as we’ve seen before an example of this object class would be OFD, ATD or PSD.

Binary fact type *[Shape Type] is allowed in [Diagram Type]* – This Binary fact type validates the usage of shapes in a diagram, this and *[Connector Type] is allowed in [Diagram Type]* are two very important steps to have a clear stencil for each diagram. An example of this Binary fact type could be “Object Class” is allowed in “OFD”.

Binary fact type *[Connector Type] is allowed in [Diagram Type]* – This Binary fact type is nearly the same as the previous, but instead of shapes it validates the allowed connectors in a diagram. As an example we can think of “Initiator” is allowed in ATD.

Object class “Connector Type” – As shapes, connectors are also key to diagrams, they often establish a connection between the shapes, or even within a shape itself. Connector Type is the abstract representation of a connector. Some examples of connectors are “initiator”, “executor”, “reference law” and “unicity law”

Binary fact type *[Connector Type] connects [Shape Type]* – This Binary fact type represents a connection between a Connector Type and a Shape Type. For example “Initiator” connects “Actor”. This does not show the whole picture of the connection by itself but in list of all unique connectors and all their shape connections do.

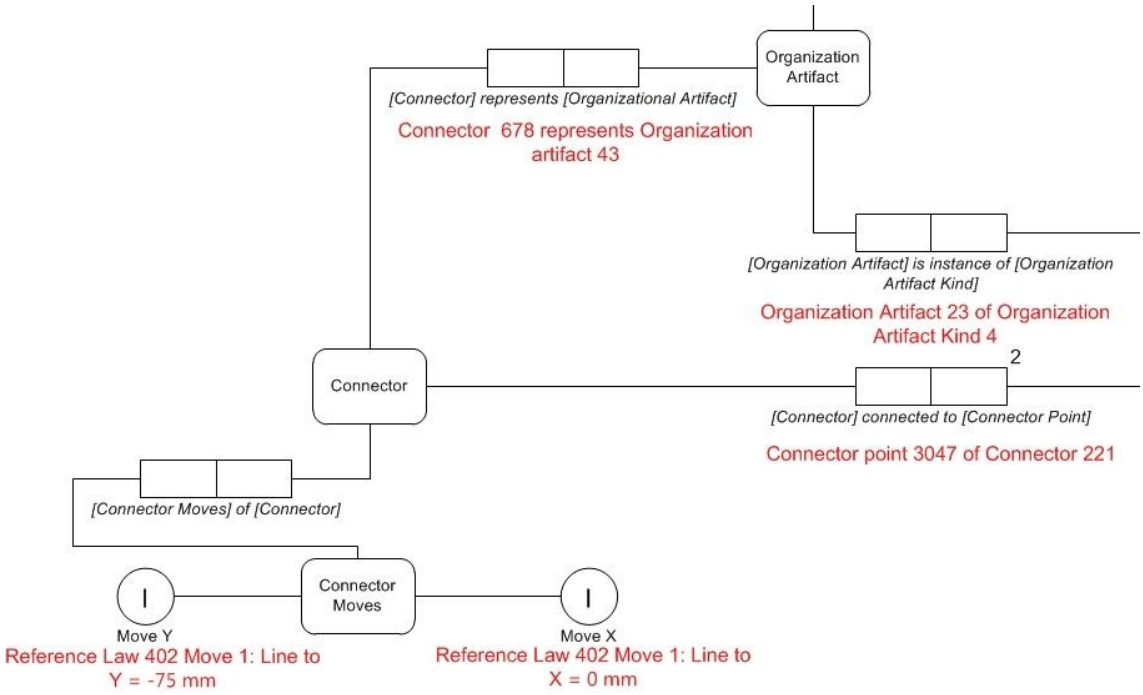


Image 36: Universal Visio Meta Model for DEMO – Part 3

Object class “Organization Artifact” – As mentioned before an organization artifact is a rule of how a shape or a connector should be used within the diagram in this case a concrete one of the diagram being referred. For example (shape) actor 183 (connector) initiator 12 (shape) transaction 97, using their label property for an easy understanding “Board” Initiates “Reduced fee approval”.

Binary fact type *[Organization Artifact] is instance of [Organization Artifact Kind]* – Same with Shapes and Shape Types, Organization Artifact are also the concrete application of an Organization Artifact Kind. For example actor 183 initiator 12 transaction 97 is instance of Actor Initiator Transaction.

Binary fact type *[Connector] represents [Organizational Artifact]* – The same as [Shape] represents [Organization Artifact] but this time for connectors. For example,

Initiates 12 (a connector of the connector type Initiator) represents actor 183 initiates 12 transaction 97.

Object class "Connector" – The concrete representation of a Connector Type, for example initiates 12, executes 36 or reference law 227.

Binary fact type *[Connector] connects to [Connector Point]* – This Binary fact type represents the connection between any connector and a particular point in a shape. In Visio this points are named Connector Point and are placed in specific places within the shapes. This act of linking the connector to a specific connector point is needed so Visio keeps track of the relation between the two and we don't have to rely on coordinates. An example of this would be initiates 12 connects to Connector Point 2 (being this Connector Point 2 the 2nd connector point of the actor Student).

Binary fact type [*Connector Moves] of [Connector]* – In a diagram is often not possible to make connections using only straight lines, so there is the need to represent the movements that the connector does while on his path to the target shape. An example of this would be Move 1 of initiates 12 (being that move one Move X: 32mm Move Y: 0mm)

Object class "Connector Moves" – As just mentioned before this class represents the changes in the connector path while reaching its destination shape. This connector moves are composed by two coordinates, a X point and a Y point both of them relative to either the previous move or the starting connector point (incase it's the first move).

Interval Scale "Move X" – These movements in the X axis uses an interval scale as they are related to the previous point and are not the absolute position within the diagram.

Interval Scale "Move Y" – Same as the Move X but in the Y axis.

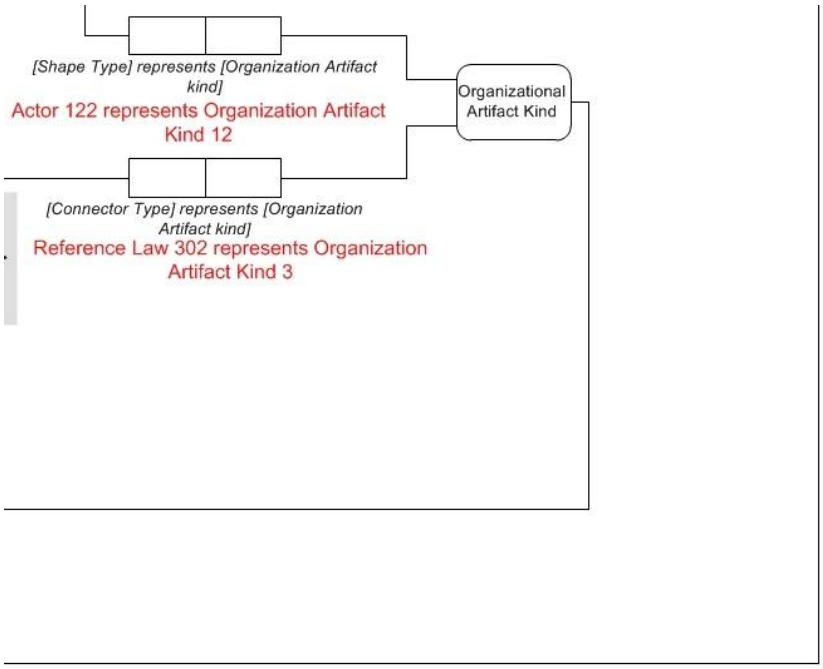


Image 37: Universal Visio Meta Model for DEMO – Part 4

Object class "Organizational Artifact Kind" – An Organizational Artifact Kind is a rule of how a shape or a connector should be used within the diagram in an abstract sense. And an example of this would be "Actor initiator Transaction" or "Actor executor Transaction".

Binary fact type *[Shape Type] represents [Organization Artifact Kind]* – Similar to [Shape] represents [Organization Artifact] but in an abstract sense. For example, transaction represents actor initiator transaction.

Binary fact type *[Connector Type] represents [Organization Artifact Kind]* – Just like [Shape Type] represents [Organization Artifact Kind] but regarding the connectors For example, Initiator represents actor initiator transaction.

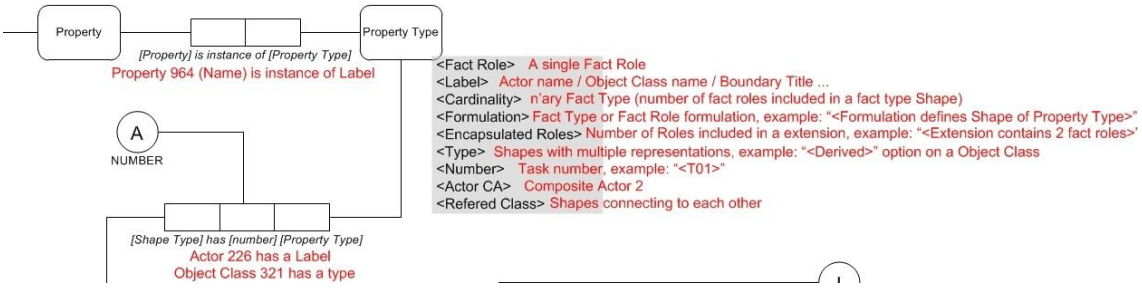


Image 38: Universal Visio Meta Model for DEMO – Part 5

Object class "Property" – Property is the concrete representation within a diagram of a property type. Each shape may have many properties and some even more then one of

the same property type. An example of a property, using our library OFD, in the object class “MONTH”, “MONTH” is a property of a property type label.

Binary fact type *[Property] is instance of [Property Type]* – All concrete properties are part of property types, the example of this Binary fact type can be “[PERSON] is member in [MEMBERSHIP]” is instance of the property type “Formulation”.

Object class “Property Type” – The abstract class of the properties. Within the DEMO diagrams there are many properties that shapes can have, such as Label, cardinality, type, encapsulated roles, or formulation.

Ternary fact type *[Shape Type] has [Number] [Property Type]* – Unlike the previous, this is a ternary relation, since each Shape Type may have from 1 to n Property Types. An example of a multiple relation could be “binary fact type has 2 fact roles”.

Absolute Scale “Number” – This Number scale only takes values within the natural numbers, but if the Fact exists it usually takes the value of 1.

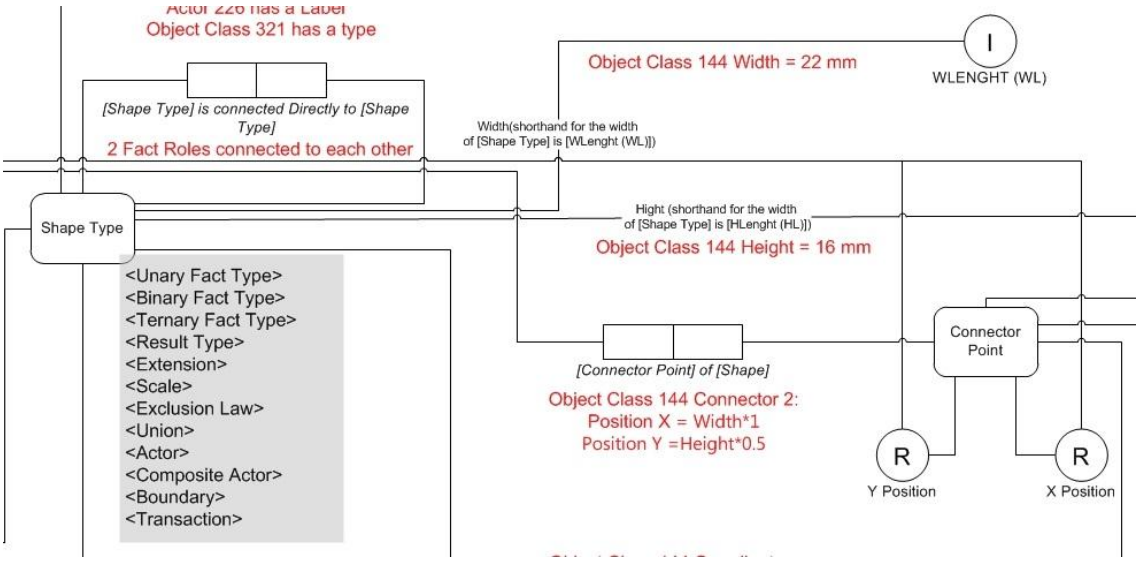


Image 39: Universal Visio Meta Model for DEMO – Part 6

Object class “Shape Type” – This object class is the abstract representation of shape. There are many shape types than can be used in DEMO such as Scale, Boundary, Actor, Transaction or Result type.

Binary fact type *[Shape Type] is connected directly to [Shape Type]* – A Shape Type may also be connected to itself. Fact roles connect directly in order to form a fact type.

Interval Scale WLENGHT (width) – Every shape type has a Width that is given with a relative value to its starting point.

Interval Scale HLENGHT (height) – Same as with width every shape type also has a height that is given with a relative value to its starting point.

Object class “Connector Point” – A connector point is a point within a Shape where a connector or another shape can be linked. Although usually defined in the stencil this connector points can be changed in their position, and in their number. In Visio they are quite easy to add, the limit to their amount is the representation size of the shape.

Binary fact type *[Connector Point] of [Shape]* – As previously mentioned a shape can have many of these connector points. An example of this fact type would be Connector Point 2 of Actor 12

Ratio Scale Y Position - The connector point Y position is obtained dividing the height for a certain value.

Ratio Scale R Position - The connector point X position is obtained dividing the width for a certain value.

Binary fact type *[Connector Point] is directly connected to [Connector Point]* – Two connector points can be linked directly. This happens in the fact Types, where Fact Roles connect to each other.

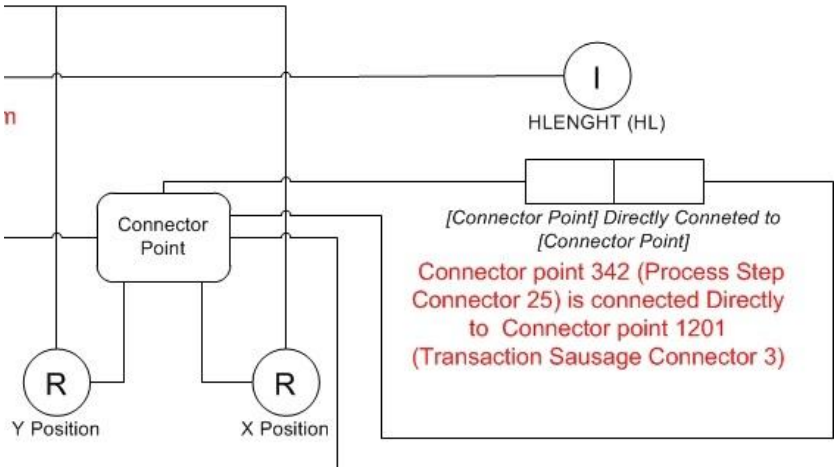


Image 40: Universal Visio Meta Model for DEMO – Part 7

3.9.3. Concrete Implementation of UVMMD on a diagram

With the model done, the next logical step was to validate such model with a concrete testing, so an access database was created to implement UVMMD and all its relations. All object classes and Scales gave place to tables and all Binary/Ternary fact type to relation tables.

In order to fill these tables we used the output of our example ATD diagram made in Visio.

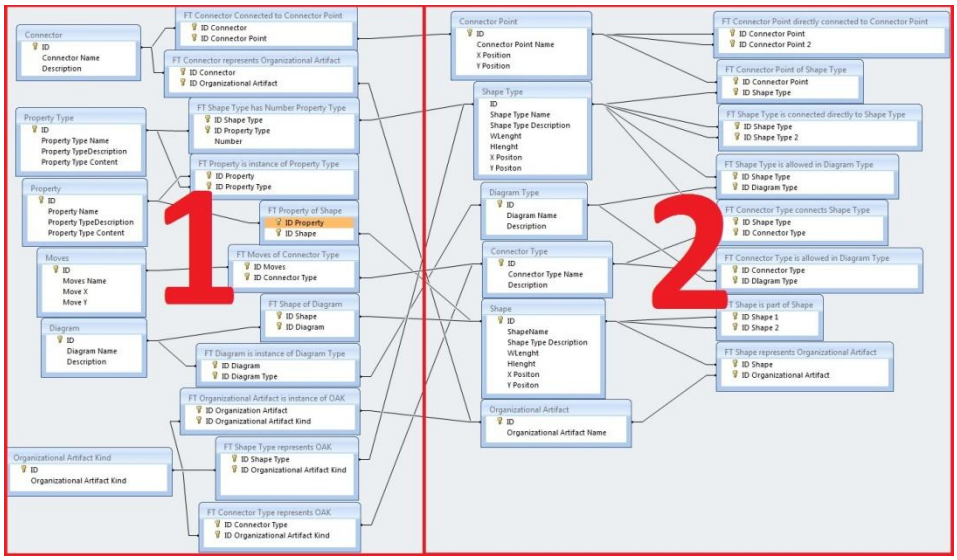


Image 41: E-R Model of Access UVMMD implementation

The Entity Relation model for our UVMMD ended up with some complexity still we decided to place it in this thesis to show how it was all related and fitting into the right places.

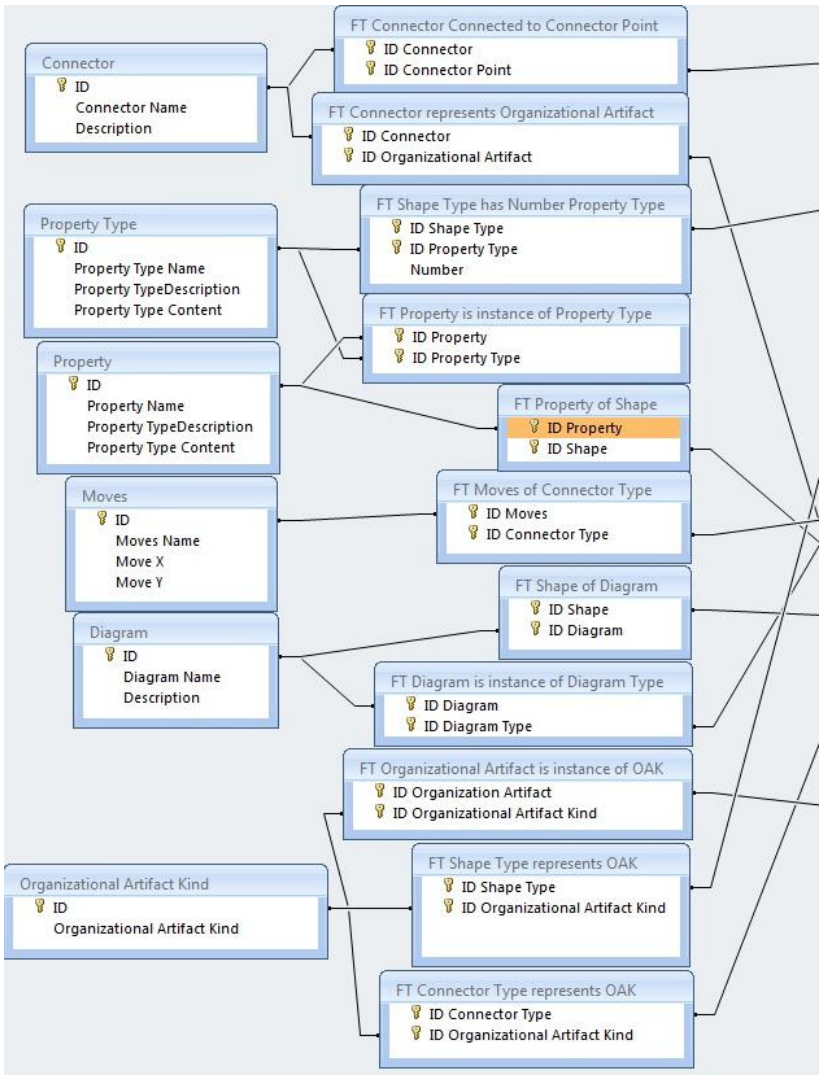


Image 42: E-R Model of Access UVMMD implementation - Part 1

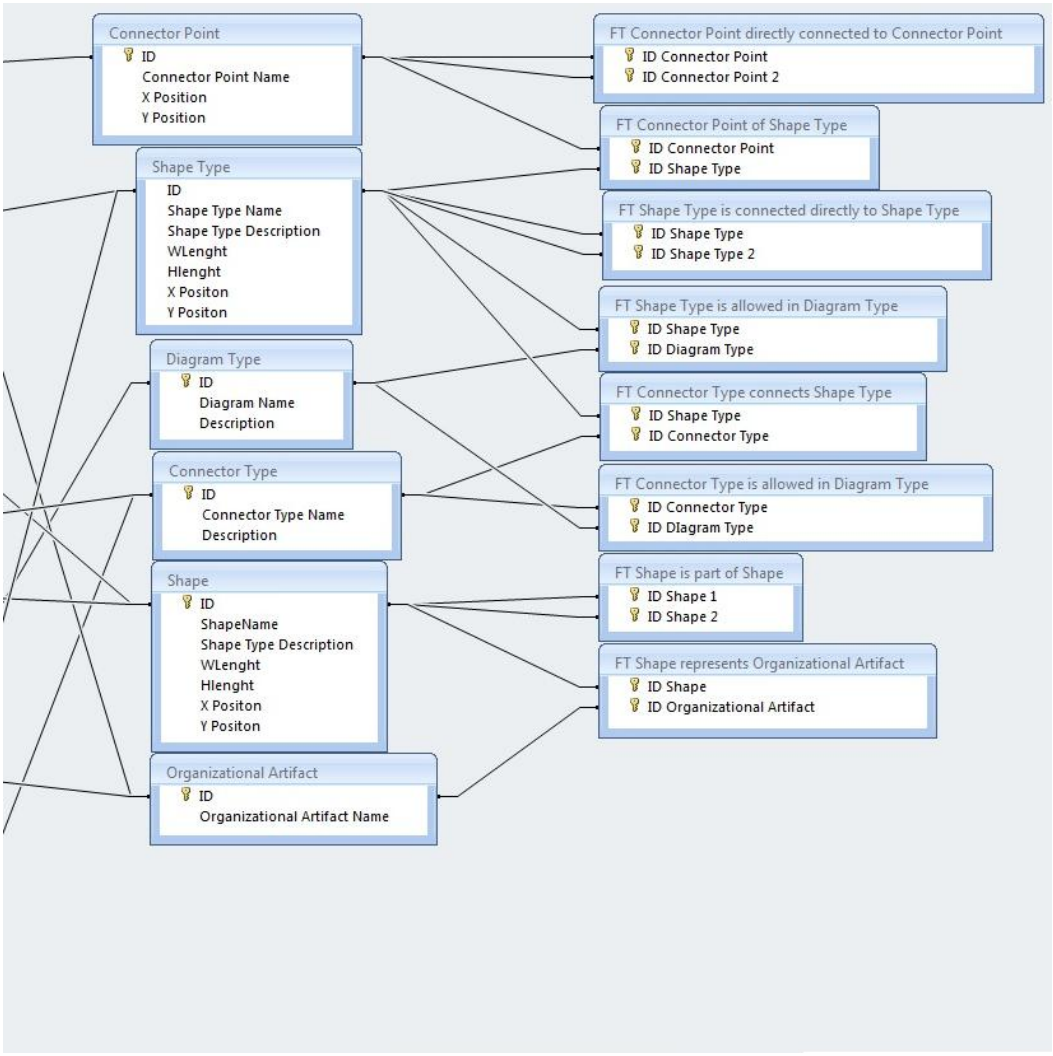


Image 43: E-R Model of Access UVMMD implementation - Part 2

3.9.4. Visio VBA solution

Having a notion of the needed information to create our Semantic Media Wiki pages and latter possibly replicate the diagrams it was time to explore Visio and VBA on the ability export the diagram information.

The first problem to solve here was what fields we needed to access to extract information. These fields and many other options like macro creation are hidden by default, so we had to make them visible. To do it, we go to “File” then “Options” and on the “Customize Ribbon” tab we activate the developer option.

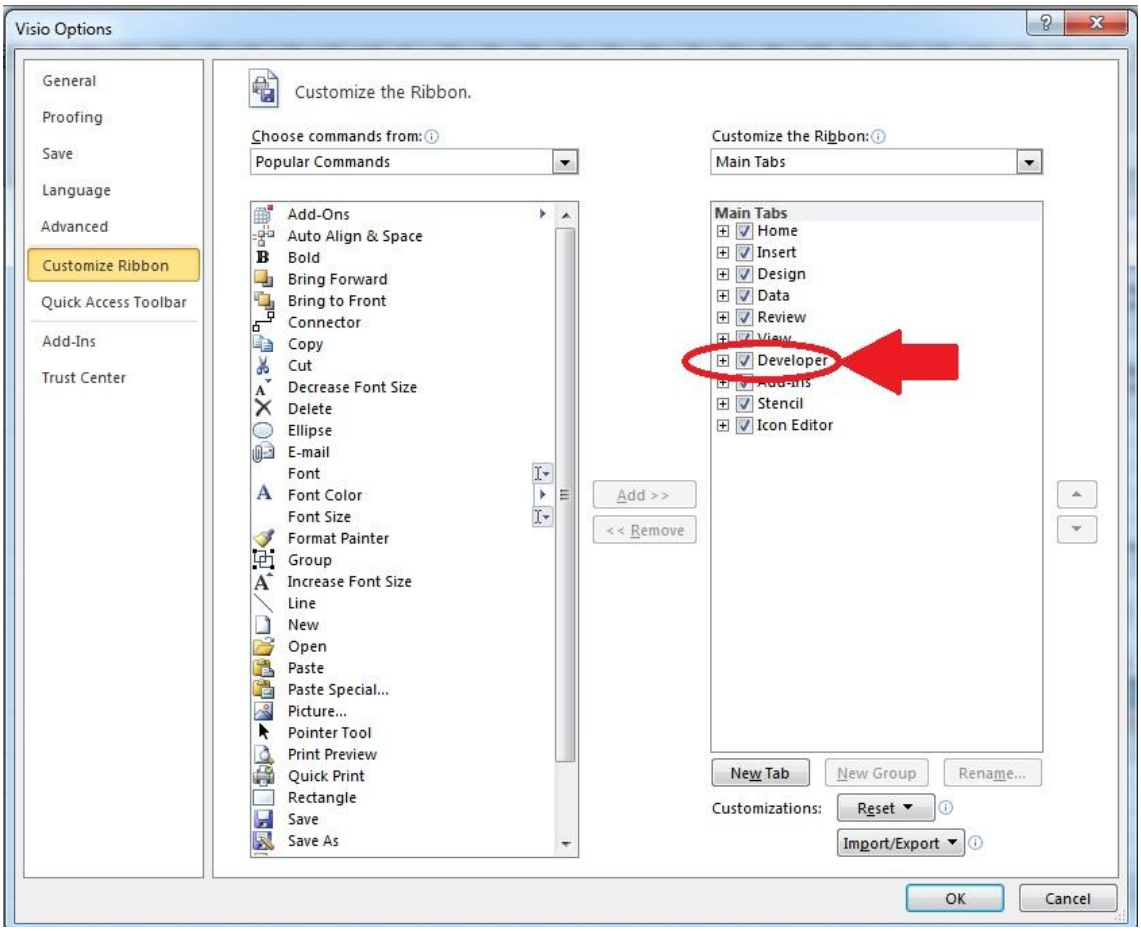


Image 44: Visio Options – Activating Developer tools

With the developer tools activated we can now access to the shape sheet for each diagram element (shape or connector). To do so we right click on the shape and select shape sheet.

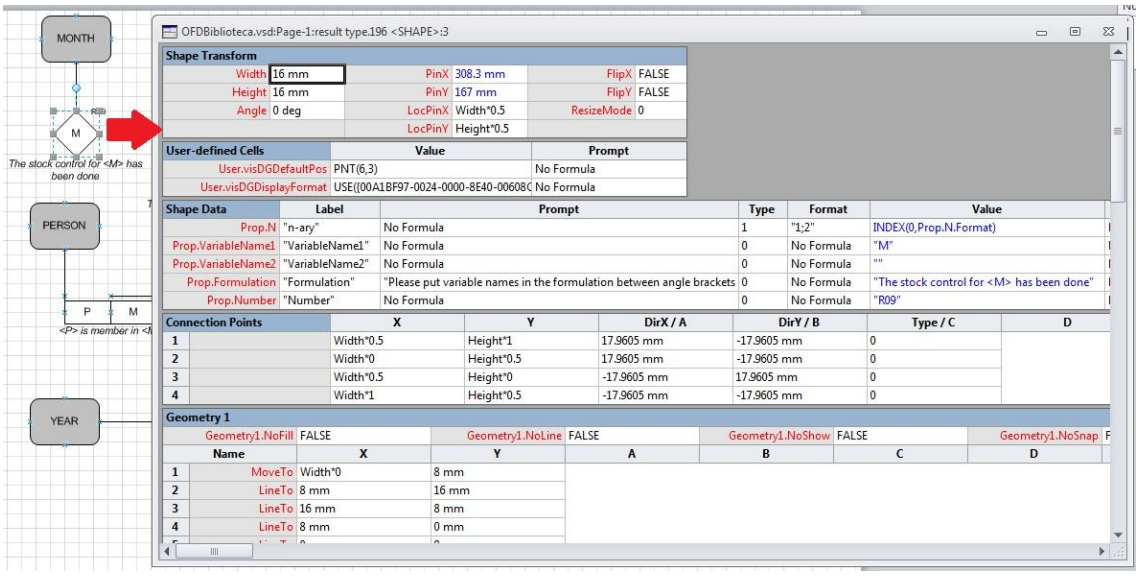


Image 45: Visio Shape Sheet of the Result Type "The stock control for <M> has been done"

In this Shape Sheet, we can access every property of the selected element. Some cells have red labels with the property names, but others don't. So to know a name of a specific cell you can select any other cell in the page and then click on the one you wish to know the name. The cell name will be written in the one you have selected. When done, you can just click escape to leave without saving.

Correspondency Table		
ATD's WOSL	**Visio Property**	**Shapes Where it's Applied**
Name	Prop.Name	Actor; Composite Actor; Transaction;
Element	shpObj.Name	All
Number	Prop.Number	Actor; Composite Actor; Transaction; Boundary;
X Position	PinX	Actor; Composite Actor; Transaction; Boundary;
Y Position	PinY	Actor; Composite Actor; Transaction;
Connector (X axis)	Connections.X* (For * >= 1 to *<= Last Connection)	Actor; Composite Actor; Transaction;
Connector (Y axis)	Connections.Y* (For * >= 1 to *<= Last Connection)	Actor; Composite Actor; Transaction;
Width	Width	Actor; Composite Actor; Transaction; Boundary;
Height	Height	Actor; Composite Actor; Transaction; Boundary;
Move X	Geometry1.X* (For * > 1 to *< Last move)	Initiator; Executor;
Move Y	Geometry1.Y* (For * > 1 to *< Last move)	Initiator; Executor;
Begining X	BeginX	Initiator; Executor;
End X	EndX	Initiator; Executor;
Actor CA	Prop.ActorCA.Type	Actor;
Boundary Title	Prop.BoundaryTitle	Boundary;

Image 46: Correspondence Table for ATD relevant properties / Visio Properties

Correspondence Table		
OFD's WOSL	**Visio Property**	**Shapes Where it's Applied**
Name	Prop.Name	Object Class; Extension; Categorial Scale; Ordinal Scale; Interval Scale; Ratio Scale; Absolute Scale;
Element	shpObj.Name	All
Number	Prop.Number	Fact Type; Result Type;
X Position	PinX	Fact Type; Result Type; Object Class; Extension; Categorial Scale; Ordinal Scale; Interval Scale; Ratio Scale; Absolute Scale; Exclusion Law; Union;
Y Position	PinY	Fact Type; Result Type; Object Class; Extension; Categorial Scale; Ordinal Scale; Interval Scale; Ratio Scale; Absolute Scale; Exclusion Law; Union;
Connector (X axis)	Connections.X* (For * >= 1 to *<= Last Connection)	Fact Type; Result Type; Object Class; Extension; Categorial Scale; Ordinal Scale; Interval Scale; Ratio Scale; Absolute Scale; Exclusion Law; Union;
Connector (Y axis)	Connections.Y* (For * >= 1 to *<= Last Connection)	Fact Type; Result Type; Object Class; Extension; Categorial Scale; Ordinal Scale; Interval Scale; Ratio Scale; Absolute Scale; Exclusion Law; Union;
Width	Width	Fact Type; Result Type; Object Class; Extension; Categorial Scale; Ordinal Scale; Interval Scale; Ratio Scale; Absolute Scale; Exclusion Law; Union;
Height	Height	Fact Type; Result Type; Object Class; Extension; Categorial Scale; Ordinal Scale; Interval Scale; Ratio Scale; Absolute Scale; Exclusion Law; Union;
Move X	Geometry1.X* (For * > 1 to *< Last move)	Reference Law; Dependency Law;
Move Y	Geometry1.Y* (For * > 1 to *< Last move)	Reference Law; Dependency Law;
Begining X	BeginX	Reference Law; Dependency Law; Unicity Law;
End X	EndX	Reference Law; Dependency Law; Unicity Law;
Type	Prop.Type.Format	Fact Type; Object Class;
Cardinality	n-aryProp.N	Fact Type; Result Type; Extension;
Variable Name 1	Prop.VariableName1	Result Type;
Variable Name 2	Prop.VariableName2	Result Type;
Formulation	Prop.Formulation	Result Type;
Fact RoleName 1	Prop.FactRoleName1	Fact Type;
Fact RoleName 2	Prop.FactRoleName2	Fact Type;
Fact RoleName 3	Prop.FactRoleName3	Fact Type;
Fact RoleName 4	Prop.FactRoleName4	Fact Type;
Fact RoleName 5	Prop.FactRoleName5	Fact Type;
Fact Formulation	Prop.FactRoleName6	Fact Type;
Encapsulated Roles	Prop.N	Fact Type;

Image 47: Correspondence Table for ATD relevant properties / Visio Properties

To help with this process, correspondence tables were made with properties we wished to export for each diagram, what name it had in vision, and in what shapes was it relevant.

At this point we had the properties and the property names to access them, but were still lacking a way to export that information.

When we activated the developer tools a new tab was unlocked in Visio, if we clicked that tab we had access to Macros and other option. Then we needed to create a new macro (the name is not relevant).

After creating a macro, a new window appears, for the Microsoft Visual Basic for Applications. In that window, at the module of our created macro we can create our exported file.

We decided our export file would be a *.txt. When populating this file we also decided to separate each shape or connector with the label "Element".

To access most of the properties we used simple if cycles, that checked the existence of the property and if true stored it at our result array OrderInfo where we store all elements.

```
If shpObj.CellExists("Prop.Name", visExistsLocally) Then
    Set celObj = shpObj.Cells("Prop.Name")
    OrderInfo(1, i - 1) = celObj.ResultStr("")
End If


'recolhe o valor da propriedade Number

If shpObj.CellExists("Prop.Number", visExistsLocally) Then
    Set celObj = shpObj.Cells("Prop.Number")
    OrderInfo(2, i - 1) = celObj.ResultStr("")
End If
```

Image 48: Accessing properties and storing in OrderInfo (example)

To access properties with multiple values we used a do while cycle. At this first solution we just worried about exporting the information, its handling was left to be done with the PHP parser. The exportation was also a simple print process.

```
For i = 0 To pagObj.Shapes.Count - 1

        Print #1, "Element"; i  ' Elemento

        If OrderInfo(1, i) <> "" Then
            Print #1, "Name: "; OrderInfo(1, i) ' Info
        End If

        If OrderInfo(2, i) <> "" Then
            Print #1, "Number: "; OrderInfo(2, i) ' Info
        End If
```

Image 49: VBA – Printing to a File (example)

The result of this process was a *.txt file with the following format for each element:

Element 3
Type: composite actor.61
Name: Publisher
Number: CA03
Position X: 15 mm
Position Y: 155 mm
WidthShape: 16 mm
HeightShape: 16 mm
ConnectorX 1: Width*0,5
ConnectorY 1: Height*0
ConnectorX 2: Width*1
ConnectorY 2: Height*0,5
ConnectorX 3: Width*0,5
ConnectorY 3: Height*1
ConnectorX 4: Width*0
ConnectorY 4: Height*0,5

This file was then going to be handled by the PHP parser.

3.9.5. Visio PHP parser

To deal with the *.txt file in PHP the first solution found was a chain of if then else conditions that looked for the specific field in the file.

```
38  $fh = fopen($myFile, 'r');
39
40  while (!feof($fh)){
41      $theData = fgets($fh);
42
43
44      $pos = strpos($theData, "Element");
45      if ($pos === false) {
46          $pos = strpos($theData, "Type");
47          if ($pos === false) {
48              $pos = strpos($theData, "Name");
49              if ($pos === false) {
50                  $pos = strpos($theData, "Number");
51                  if ($pos === false) {
52                      $pos = strpos($theData, "ActorCA");
53                      if ($pos === false) {
54                          $pos = strpos($theData, "BoundaryName");
55                          if ($pos === false) {
56                              $pos = strpos($theData, "Cardinality");
57                              if ($pos === false) {
58                                  $pos = strpos($theData, "Formulation");
59                                  if ($pos === false) {
60                                      $pos = strpos($theData, "Variable");
61                                      if ($pos === false) {
```

Image 50: PHP Parser – Searching for Keywords (example)

When this word was found it was then stored, depending on the shape (for example actor) in an array with that name.

```
            }
        }
    }else{
        $content = explode("Name:", $theData);
        $result["Name $forma"] = $content[1];
        if ($content[1] != ""){
            if ($tipo == actor){
                $actor["Name $forma"] = $content[1];
            } else {
                if ($tipo == transaction){
                    $transaction["Name $forma"] = $content[1];
                } else {
                    if ($tipo == connector){
                        $connector["Name $forma"] = $content[1];
                    } else {
                        if ($tipo == boundary){
                            $boundary["Name $forma"] = $content[1];
                        } else {
                            if ($tipo == facttype){
                                $facttype["Name $forma"] = $content[1];
                            } else {
                                if ($tipo == resulttype){
                                    $resulttype["Name $forma"] = $content[1];
                                } else {
                                    if ($tipo == objectclass){
                                        $objectclass["Name $forma"] = $content[1];
                                    } else {
                                        if ($tipo == extension){
                                            $extension["Name $forma"] = $content[1];
                                        } else {
                                            if ($tipo == scale){
                                                $scale["Name $forma"] = $content[1];
                                            } else {
                                                if ($tipo == law){
                                                    $law["Name $forma"] = $content[1];
                                                }
                                            }
                                        }
                                    }
```

Image 51: PHP Parser – Storing Data (example)

Besides these two steps there was also the need to treat specific fields for each element such as the connector points that were given based on the shapes height and width. Such corrections were being done calling functions before storing the data.

Some issues were found while working on this solution, like the use of "<" ">" tags in Result Type and fact type formulations caused them to be acknowledge as HTML tags, or that Visio used comas to mark the decimal values while PHP uses dots.

These two problems had straight forward solutions, the "<" ">" tags were replaced for "[" "]" and the comas for dots using functions in PHP.

The output of this parser after this treatment was similar to what with had from the VBA but now with all the calculations solved and issues treated. Using the same example, now an element would look like:

Element 3
Visio Name: composite actor.61
Name: Publisher
Number: CA03
Position X: 15

Position Y: 155
Width: 16
Height: 16
ConnectorX 1: 8
ConnectorY 1: 0
ConnectorX 2: 16
ConnectorY 2: 8
ConnectorX 3: 8
ConnectorY 3: 16
ConnectorX 4: 0
ConnectorY 4: 8

But this was still far from a Semantic Media Wiki Page, so we had the need to now learn how such pages were to be created and how to automate the process.

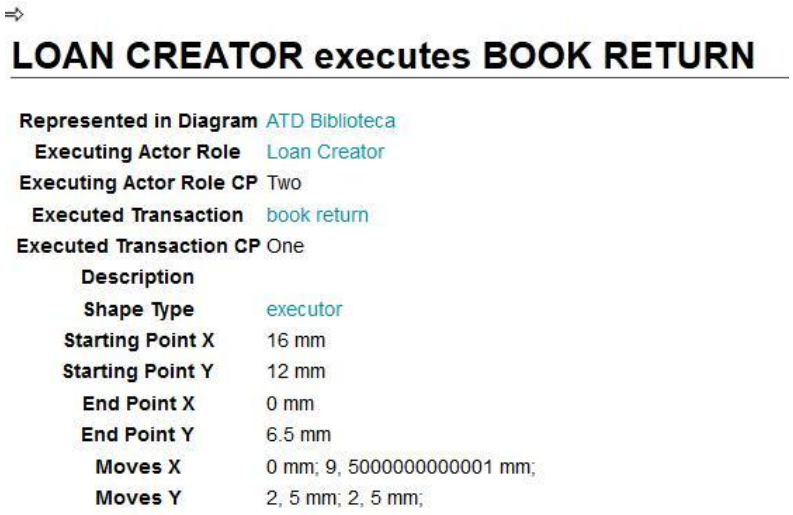
3.10. Semantic Media Wiki Implementation

In order to properly install Semantic Media Wiki to work with DEMO we followed the procedures described in an installation guide available in J. Capela thesis entitled *Semantic Media Wiki adaptation to support Organizational Engineering*[15] and transcribed in full in the appendix section.

Due to malfunctions between programs, and support issues, the installation only works for specific versions of the programs.

The software used was Xampp 1.7.1 for windows, Mediawiki 1.15.3, Semantic Media Wiki 1.5.1_1 and semantic forms 2.0.9. The tutorial involves more software, but those were not required for our task at hand.

To test the proper installation of all the components, some test pages for our diagram elements were created with the properties that we had already ruled as important for a coherent recreation of the diagram without data loss.



LOAN CREATOR executes BOOK RETURN

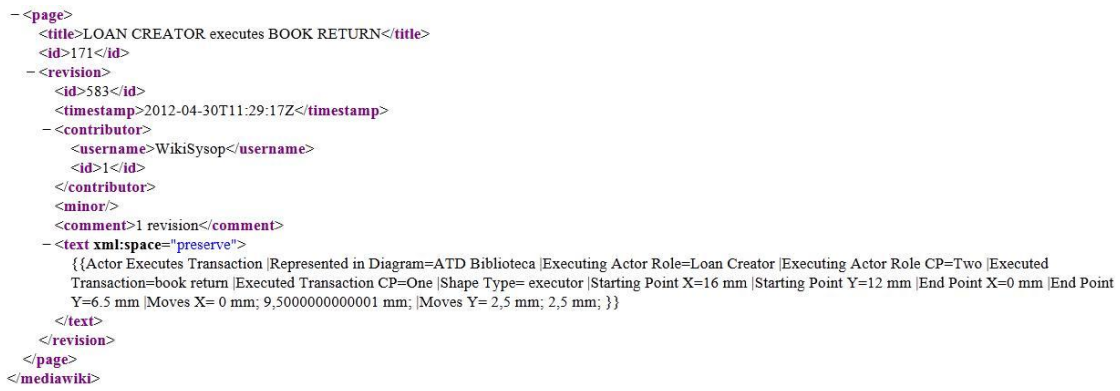
Represented in Diagram	ATD Biblioteca
Executing Actor Role	Loan Creator
Executing Actor Role CP	Two
Executed Transaction	book return
Executed Transaction CP	One
Description	
Shape Type	executor
Starting Point X	16 mm
Starting Point Y	12 mm
End Point X	0 mm
End Point Y	6.5 mm
Moves X	0 mm; 9, 5000000000001 mm;
Moves Y	2, 5 mm; 2, 5 mm;

Image 52: Semantic Media Wiki Page – Loan Creator executes Book Return

With everything properly installed and tested following the installation guide our next task was to figure how we could import all elements into semantic media wiki.

The starting idea was to try to import directly into SMW SQL database. This idea didn't last long, due to the fact that pages are not stored together but instead have many components spread throughout the many tables that compose the database.

An easier solution was needed, so with a few example pages created we tried the built in export tool, that creates a *.XML version of the page at hand.



```
- <page>
    <title>LOAN CREATOR executes BOOK RETURN</title>
    <id>171</id>
  - <revision>
      <id>583</id>
      <timestamp>2012-04-30T11:29:17Z</timestamp>
    - <contributor>
        <username>WikiSysop</username>
        <id>1</id>
      </contributor>
      <minor/>
      <comment>1 revision</comment>
    - <text xml:space="preserve">
        {{Actor Executes Transaction |Represented in Diagram=ATD Biblioteca |Executing Actor Role=Loan Creator |Executing Actor Role CP=Two |Executed
        Transaction=book return |Executed Transaction CP=One |Shape Type= executor |Starting Point X=16 mm |Starting Point Y=12 mm |End Point X=0 mm |End Point
        Y=6.5 mm |Moves X= 0 mm; 9,5000000000001 mm; |Moves Y= 2,5 mm; 2,5 mm; }}
      </text>
    </revision>
  </page>
</mediawiki>
```

Image 53: Semantic Media Wiki – Export of Loan Creator executes Book Return

This XML page was a more plausible solution to be recreated in PHP, and with a built in import tool also available in SMW it seemed like a perfect fit for our needs. But first another test had to be made. Individually importing every single page would be an amazingly time consuming and repetitive task for large sized diagrams, so the possibility to include all pages in a single file had to be tested.

Multiple SMW pages in a single file were in fact possible and the importation process was created or updated without issues.

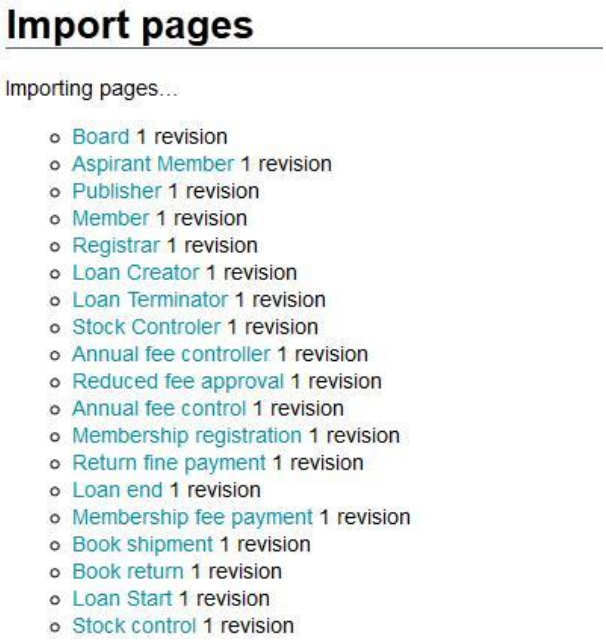


Image 54: Semantic Media Wiki – Multiple Pages Importation

Now that we knew we had a base to work on, we changed our focus to the structure needed to make sure our SMW pages were coherent and kept their structure. To do so we, again, based ourselves in Jorge Capela's *Semantic Media Wiki adaptation to support Organizational Engineering.*

The first step to keep everything coherent was to create a semantic property for every single property we would use. These properties allowed us to define the data types allowed in those specific fields, making sure that not only the data type was correct, but also that the semantic part of Semantic Media Wiki would in fact work.

Image 55: SMW – Creating a property

This is also a mandatory step before we are able to create templates.

Templates in SMW are basically page structures. By creating a template and forcing a page to comply with it, we can make sure that all similar elements of a diagram will share similar pages and we set the boundary for what fields are really needed for that element.

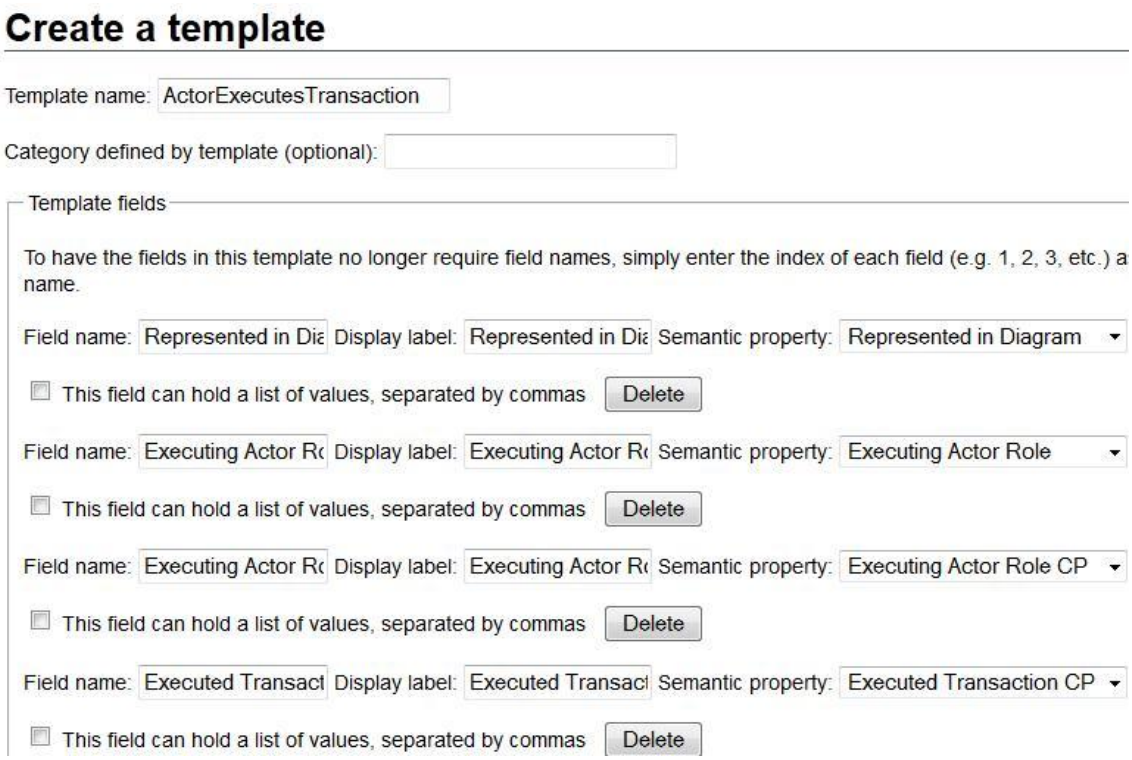


Image 56: SMW – Creating a Template

In the template creations we can also define the display label for each category. For the sake of simplicity we kept the same names. Another available option is the possibility or not for the field to hold multiple values, in our case we will require some to fields to do so, like the connector points fields and the connector movements fields.

The final step of the process in defining a structure is the creation of forms. To create a form you need to have created templates and you can chose the one you wish to apply in that specific form. In the form creation you can still alter the labels that will be presented when creating a page, and you can also select if any specific field is mandatory (needs to be filled) hidden (not visible to regular users) or restricted (only administrators can fill it).

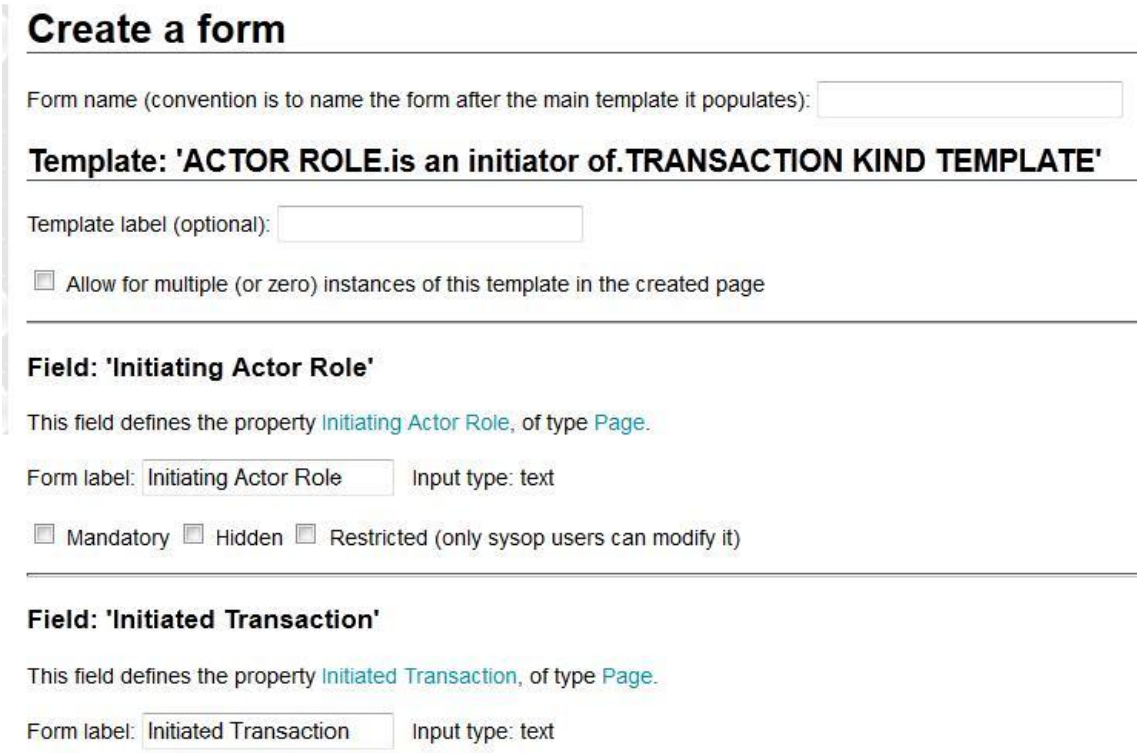


Image 57: SMW – Creating a Form

After creating forms for all templates all is ready to start the creation of pages. Although possible to do in SMW as obvious, our objective was to create the pages in the PHP parser, so our attention was turned in that direction again.

3.11. PHP Parser Output

With the current implementation we had on the PHP parser, the creation of the XML file was straightforward. In every SMW pages there are many lines of XML code that were the same, so those were made into a function and called every time a new page started.

There was also the need of data that the Diagram exportation file does not provide, like the diagram name, the actor name, a diagram description (not mandatory) and a timestamp. The timestamp was solved with an inbuilt function of the PHP for that, and for the rest, a layout page was created to allow the user to enter his data, and also to select where the input *.txt file that the parser was going to work on was located.

Finally to fill the information about each element we used a similar solution as the one already being used to classify the elements but this time that only searched on the specific array we were working at.

```php
if (empty($transaction)){
} else{
    foreach($transaction as $key => &$value)
    {
        $pos = strpos($key, "Element");
        if ($pos === false) {
            $pos = strpos($key, "Name");
            if ($pos === false) {
                $pos = strpos($key, "Number");
                if ($pos === false) {
                    $pos = strpos($key, "PositionX");
                    if ($pos === false) {
                        $pos = strpos($key, "PositionY");
                        if ($pos === false) {
                            $pos = strpos($key, "Width");
                            if ($pos === false) {
                                $pos = strpos($key, "Height");
                                if ($pos === false) {
                                } else{
                                $height = rtrim($value);}
                            } else{
                                $width = rtrim($value);}
                        } else{
                            $representedinY = rtrim($value);}
                    } else{
                        $representedinX = rtrim($value);}
                } else{
                    $transactionID = rtrim($value);}
            } else{
                $transactionName = rtrim($value);}
        } else {

            $factType = rtrim(nonumbers($value));

            echo "<br>";
            Echo "$key => $value";
            echo "<br>";
            $pageName = rtrim(ltrim($transactionName));
            if (empty($pageName) == False)
```

Image 58: PHP Parser – Extracting data for the XML

With the information gathered, all left to do was to write it in to the file respecting the Templates that were created.

```
                fwrite($fh, $stringData);
                $stringData= '<text xml:space="preserve">{{TRANSACTION KIND TEMPLATE';
                fwrite($fh, $stringData);
                $stringData= "\n|Transaction ID=$transactionID\n";
                fwrite($fh, $stringData);
                $stringData= "|Transaction Name=$transactionName\n";
                fwrite($fh, $stringData);
                $stringData= "|Transaction Description=$transactionDescription\n";
                fwrite($fh, $stringData);
                $stringData= "|Fact type=$factType\n";
                fwrite($fh, $stringData);
                $stringData= "|Represented in X=$representedinX\n";
                fwrite($fh, $stringData);
                $stringData= "|Represented in Y=$representedinY\n";
                fwrite($fh, $stringData);
                $stringData= "|Height=$height\n";
                fwrite($fh, $stringData);
                $stringData= "|Width=$width";
                fwrite($fh, $stringData);
                $stringData= "\n}}</text>\n</revision>\n</page>\n\n";
                fwrite($fh, $stringData);
            }
        }
    }
```

Image 59: PHP Parser – Writing data on XML

Although now having a solution to our initial problem, we realized it was probably not the most appropriate and that some further steps could be taken to make it better.

3.12. A Expandable Solution

In order to facilitate the future incorporation of developments in the DEMO methodology (that is still on its first steps) we decided that we would try to approach the solution from another perspective so that future new diagrams or alterations to the current ones were easier to incorporate.

3.12.1. Formatting in VBA

The first step to this new solution was to do all the properties formatting in the VBA instead of in Visio. Since the VBA code would be forever bound to the diagram at hand, and we couldn't change that, we decided that there was the best place to treat the output file allowing this way for a PHP parser abstract enough to treat any kind of diagram.

One of the first things that needed solving was picking the beginning shape and the end shape on each connector. This may seem irrelevant, but it has great importance at a later stage to define the SMW page names properly. If we didn't do such defining we could, for example, end up with transactions that "initiated" actors, and we wouldn't want that to happen.

The easiest solution found to accomplish this was to study the shapes. After this study it was possible to acknowledge that some shapes could always be considered the beginning shape and others always the ending shape.

There was also a related case, that was very specific, the "unicity law", in this case there is no end shape, and they are both the same.

To solve both issues we opted for a chain of "If in string" comparisons and depending on the case printed the appropriate result. There was also the need to exclude part of the clutter in the line where the connector point of the connected shape was presented, for that we used replaces of the text for empty space (these text is always the same regardless of the connector or shape).

```
If OrderInfo(221, i) <> "" Then
    aux = Trim(Replace(OrderInfo(221, i), "_XFTRIGGER(", ""))
    aux = Trim(Replace(aux, "!EventXFMod)", ""))
    If InStr(OrderInfo(221, i), "transaction") >= 1 Then Print #1, "End Shape: "; aux ' Info
    If InStr(OrderInfo(221, i), "actor") >= 1 Then Print #1, "Begining Shape: "; aux ' Info
    If InStr(OrderInfo(221, i), "fact") >= 1 Then Print #1, "Begining Shape: "; aux ' Info
    If InStr(OrderInfo(221, i), "result") >= 1 Then Print #1, "Begining Shape: "; aux ' Info
    If InStr(OrderInfo(221, i), "scale") >= 1 Then Print #1, "Begining Shape: "; aux ' Info
    If InStr(OrderInfo(221, i), "object") >= 1 Then Print #1, "End Shape: "; aux ' Info
    If InStr(OrderInfo(221, i), "extension") >= 1 Then Print #1, "End Shape: "; aux ' Info
End If

If OrderInfo(3, i) <> "" Then
    aux6 = "unicity law"
    If InStr(OrderInfo(0, i), aux6) = 1 Then
        aux6 = Trim(Replace(OrderInfo(3, i), "PAR(PNT(", ""))
        aux6 = Trim(Replace(aux6, "!onnections.X", ""))
        aux6 = Trim(Replace(aux6, "!onnections.Y", ""))
        aux6 = Trim(Replace(aux6, "))", ""))
        aux6 = Trim(Replace(aux6, ";", ""))
        aux6 = Mid(aux6, 1, 13)
        aux6 = Trim(Replace(aux6, "!", ""))
        aux6 = Trim(Replace(aux6, "C", ""))
        Print #1, "Begining Shape: "; aux6 ' Info
        aux7 = Trim(Replace(OrderInfo(3, i), "PAR(PNT(", ""))
        aux7 = Trim(Replace(aux7, "!Connections.", ""))
        aux7 = Trim(Replace(aux7, "!Connections.", ""))
        aux7 = Trim(Replace(aux7, "))", ""))
        aux7 = Trim(Replace(aux7, "fact type", ""))
        Length = Len(aux7)
        aux7 = Mid(aux7, (Length - 1), 3)
        aux7 = Trim(Replace(aux7, "Y", ""))
        Print #1, "Begining Shape CP: "; aux7 ' Info
    End If


    If InStr(OrderInfo(0, i), "unicity law") = 0 Then
        aux2 = Trim(Replace(OrderInfo(3, i), "PAR(PNT(", ""))
        aux2 = Trim(Replace(aux2, aux, ""))
        aux2 = Trim(Replace(aux2, "!Connections.X", ""))
        aux2 = Trim(Replace(aux2, "!Connections.Y", ""))
        aux2 = Trim(Replace(aux2, "))", ""))
        aux2 = Trim(Replace(aux2, ";", ""))
        aux2 = Mid(aux2, 1, 1)
        If InStr(OrderInfo(221, i), "transaction") >= 1 Then Print #1, "End Shape CP: "; aux2 ' Info
        If InStr(OrderInfo(221, i), "actor") >= 1 Then Print #1, "Begining Shape CP: "; aux2 ' Info
    If InStr(OrderInfo(221, i), "fact") >= 1 Then Print #1, "Begining Shape CP: "; aux2 ' Info
    If InStr(OrderInfo(221, i), "result") >= 1 Then Print #1, "Begining Shape CP: "; aux2 ' Info
    If InStr(OrderInfo(221, i), "scale") >= 1 Then Print #1, "Begining Shape CP: "; aux2 ' Info
    If InStr(OrderInfo(221, i), "object") >= 1 Then Print #1, "End Shape CP: "; aux2 ' Info
    If InStr(OrderInfo(221, i), "extension") >= 1 Then Print #1, "End Shape CP: "; aux2 ' Info
     End If
End If
```

Image 60: VBA – Printing Beginning/End shape and its connector point (one side of the connector)

Another property that needed formatting was the width. Unlike height that is give for every shape, width varies in the fact types, so that variation needed to be considered.

```
If OrderInfo(20, i) <> "" Then
    aux = Trim(Replace(OrderInfo(20, i), """", ""))
    If IsNumeric(Trim(aux)) Then Print #1, "EncapsulatedRoles: "; aux ' Info
End If

If OrderInfo(7, i) <> "" Then
     aux = Trim(Replace(OrderInfo(7, i), "mm", ""))
     If IsNumeric(Trim(aux)) Then realwidth = CDbl(aux)
     If IsNumeric(Trim(aux)) Then Print #1, "Width: "; realwidth
     If Not IsNumeric(Trim(aux)) Then
         aux = Trim(Replace(OrderInfo(7, i), "cm*Prop.N", ""))
         If IsNumeric(Trim(aux)) Then aux3 = CDbl(aux)
         aux2 = Trim(Replace(OrderInfo(20, i), """", ""))
         If IsNumeric(Trim(aux2)) Then aux4 = CDbl(aux2)
         aux5 = (aux3 * 10) * aux4
         realwidth = aux5
         Print #1, "Width: "; realwidth
     End If
```

Image 61: VBA – Printing Encapsulated Roles and calculating Width

Most of the other variables were straightforward with the occasional need for some clutter removal. With all the formatting done, an element had the following appearance:

```
19  Element 1
20  Label: The stock control for [M] has been done
21  Visio Name: result type.196
22  Number: R09
23  Position X: 308,3
24  Position Y: 167
25  Width:  16
26  Height: 16
27  Formulation: The stock control for {M] has been done
28  Variable 1: M
29  ConnectorX 1:  8
30  ConnectorY 1:  16
31  ConnectorX 2:  0
32  ConnectorY 2:  8
33  ConnectorX 3:  8
34  ConnectorY 3:  0
35  ConnectorX 4:  16
36  ConnectorY 4:  8
37  MoveY 1: 8
38  MoveX 2: 8
39  MoveY 2: 16
40  MoveX 3: 16
41  MoveY 3: 8
42  MoveX 4: 8
43  MoveY 4: 0
44  MoveX 5: 0
45  MoveY 5: 8
```

Image 62: OFD Output text file – Element

Before this new option of formatting in VBA the appearance was of the image bellow. As you can see there are a few changes, we now had the correct connector positions, the variable name, and the label that would give name to the SMW page and no longer had the encapsulated roles line that in a result type only contains clutter.

```
28  Element 1
29  Type: result type.196
30  Number: R09
31  Position X: 308,3 mm
32  Position Y: 167 mm
33  WidthShape: 16 mm
34  HeightShape: 16 mm
35  Formulation: The stock control for [M] has been done
36  EncapsulatedRoles: INDEX(0;Prop.N.Format)
37  ConnectorX 1: Width*0,5
38  ConnectorY 1: Height*1
39  ConnectorX 2: Width*0
40  ConnectorY 2: Height*0,5
41  ConnectorX 3: Width*0,5
42  ConnectorY 3: Height*0
43  ConnectorX 4: Width*1
44  ConnectorY 4: Height*0,5
45  MoveX 1: Width*0
46  MoveY 1: 8 mm
47  MoveX 2: 8 mm
48  MoveY 2: 16 mm
49  MoveX 3: 16 mm
50  MoveY 3: 8 mm
51  MoveX 4: 8 mm
52  MoveY 4: 0 mm
53  MoveX 5: 0 mm
54  MoveY 5: 8 mm
```

Image 63: OFD Output text file – Element (old version)

3.12.2. PHP parser with XML rules and parameters

With the output formatting done in the VBA step, our PHP parser could now be free of any reference to concrete diagram shapes or connectors. In order to accomplish this goal, further steps were still needed, in the parser there was still the need to know what properties each element would have, and when exporting to the XML format for the SMW pages, the template for each shape.

For each of these two problems a XML file seemed like the best solution. As it's easy to alter and rapidly integrated into the PHP code.

```
41      <ShapeType>
42          <ShapeName>transaction</ShapeName>
43          <UnaryPropertyType>
44              <PropertyName>Type:</PropertyName>
45              <PropertyName>Name:</PropertyName>
46              <PropertyName>Number:</PropertyName>
47              <PropertyName>Position X:</PropertyName>
48              <PropertyName>Position Y:</PropertyName>
49              <PropertyName>WidthShape:</PropertyName>
50              <PropertyName>HeightShape:</PropertyName>
51          </UnaryPropertyType>
52          <MultiplePropertyType>
53              <PropertyName>ConnectorX</PropertyName>
54              <PropertyName>ConnectorY</PropertyName>
55          </MultiplePropertyType>
56      </ShapeType>
57
58      <ConnectorType>
59          <ConnectorName>initiator</ConnectorName>
60          <UnaryPropertyType>
61              <PropertyName>Type:</PropertyName>
62              <PropertyName>Begining Shape:</PropertyName>
63              <PropertyName>Begining Shape CP:</PropertyName>
64              <PropertyName>End Shape:</PropertyName>
65              <PropertyName>End Shape CP:</PropertyName>
66              <PropertyName>Position X:</PropertyName>
67              <PropertyName>Position Y:</PropertyName>
68          </UnaryPropertyType>
69          <MultiplePropertyType>
70              <PropertyName>MoveX</PropertyName>
71              <PropertyName>MoveY</PropertyName>
72          </MultiplePropertyType>
73      </ConnectorType>
```

Image 64: XML Structure of ATD elements (partial)

The layout we decided to use to solve which properties each element would have, was to first, separate the diagrams. Inside those diagrams we would place two kinds of elements, “ShapeType” for the shapes and “ConnectorType”. Then with the container for each one built we needed name the element, and define its properties. We also decided it would be best to split the property’s in two different types, the unary that only have one entry for each element and the multiple that can have many entries for each element.

```
<DiagramType>
    <DiagramKind>ATD</DiagramKind>

    <ShapeType>
        <ShapeKind>actor</ShapeKind>
        <PropertyList>
            <PropertyName>"\n|Represented in Diagram=CurrentDiagram:</PropertyName>
            <PropertyName>"\n|Actor ID=Number:</PropertyName>
            <PropertyName>"\n|Actor Name=Name:</PropertyName>
            <PropertyName>"\n|Actor Description=Description:</PropertyName>
            <PropertyName>"\n|Actor Type=Type:</PropertyName>
            <PropertyName>"\n|Shape type=Visio Name:</PropertyName>
            <PropertyName>"\n|Connectors X=ConnectorX:</PropertyName>
            <PropertyName>"\n|Connectors Y=ConnectorY:</PropertyName>
            <PropertyName>"\n|Width=Width:</PropertyName>
            <PropertyName>"\n|Height=Height:</PropertyName>
            <PropertyName>"\n|Position X=Position X:</PropertyName>
            <PropertyName>"\n|Position Y=Position Y:</PropertyName>
        </PropertyList>
    </ShapeType>
```

Image 65: XML Structure of ATD's Actor SMW page

For the layout for the SMW pages creation, a similar XML was created, but in this case for each kind of page, we listed the properties in that template. One very important aspect to consider here is the naming, as the resulting name needed to be the same as the property in the previous XML file.

The way of working of the PHP parser was rather simple, the first step was to import the *txt file we got exporting the Visio diagram attributes with the VBA and store it in a array. Then we needed to store in a different array the "Visio Name" and the "Label" properties of all shapes in order to latter compare and match with the parameter we get from the connectors.

```
Element 37
Visio Name: initiator.22
Begining Shape: actor.46
Begining Shape CP: 1
End Shape: transaction.16
End Shape CP: 3
Width:  108,667971001578
MoveX 1: 0
MoveY 1: -2,1859074513241
MoveX 2: 3,75
MoveY 2: -2,1859074513241
MoveX 3: 3,75
MoveY 3: -2,8140925486759
MoveX 4: 13,479234341059
MoveY 4: -2,8140925486759
```

```
Element 9
Visio Name: actor.46
Name: Stock Controler
Number: A09
Position X: 73
Position Y: 155
Width:  16
Height: 16
ConnectorX 1:  16
ConnectorY 1:  5
ConnectorX 2:  16
ConnectorY 2:  12
ConnectorX 3:  8
ConnectorY 3:  16
ConnectorX 4:  0
ConnectorY 4:  8
ConnectorX 5:  33,0739775854768
ConnectorY 5:  12,0788658478766
```

Image 66: ATD initiator and his connected actor

This is a necessary step in order to obtain the correct connector page name.

After these steps we imported our XML file containing the shapes for all diagrams and its properties and the XML file with the SMW templates. Is of the utmost importance that those properties not only match the names given in our VBA export file, but also, the names given in our SMW template structure XML and that all diagram elements share the same names in all three, with the exception of the numbers in the txt file, that get removed for comparison.

With this files imported we could then compare the diagram names and pick the right diagram element to compare and print the correct fields to our final xml. In those elements marked as multi property type the comparison is made multiple times while that property exists within the element, and separated in the result string with ";" for a correct compliance with our SMW pages rules upon printing.

4. Future Development

4.1. Creation of Validation Rules in Visio

In most modeling software you are not limited to what your stencil allows you to use. And Visio is no exception, you can simple grab whatever shape or connector from any other stencil and integrate it in your diagram. This, in our case poses as a problem, as even if the solution is abstract enough to accept all shapes and connectors, you need to account for the ones you will be using, and not having new ones being added as you please that don't even belong in the DEMO methodology.

There is also another issue, DEMO diagrams have specific rules, for example an ATD transaction cannot start another ATD transaction, or an OFD object class cannot be directly connected to another OFD object class.

Or even on a more basic level, we have the need that all connectors are linked to something and all shape fields are filled properly.

To help with these issues Visio 2010 comes with a new validation mechanism where you can add new rules that you find necessary to validate your diagrams. Since we decided to work with Visio we feel that this is a tool that could help us in a future development not only make sure our parser works properly but also help the person modeling acknowledge and fix his mistakes at the time of doing.

4.2. Generalization of the VBA solution

One of the issues in our VBA solution is the need to consider, in most connectors, what the starting point is and what the ending point is. To do so we take in consideration the shape that is being connected in each side. Not taking this step would mean incorrect names in our SMW pages.

To generalize our VBA solution a possible solution would be a change in the connector symbols so that the user at the time of the connection can distinguish and place properly the beginning and the end points of the connector, as it happens in the ATD executer where a small black square always marks the actor end.

With possible solve to the connectors part, we are still left with the variable width of the fact types. To that a solution would be for instead of using a field named encapsulated roles, use something more general in every shape where it applies and upon its existence apply the calculations for the correct width.

4.3. Completing the Cycle

The possibility to recreate the diagrams from the SMW pages was one of the focal points in this work, and so making sure we didn't lose any bit of relevant data was a preoccupation throughout the whole process.

Unfortunately even making sure all relevant data was there at the end, recreating these diagrams in Visio would be close to impossible, and it would involve working with the *vdx file and passing all parameters, most only relevant for Visio into our SMW pages making them long, dull and not of much use since the valuable information would be lost in the middle. Adding to that, the slightest user change in the page would cause a chain effect in so many properties that fixing them all to recreate the diagram would be a colossal task.

Hopefully there is other software capable to recreate such diagrams with only the parameters we though essential, and the concept was proven in *Semantic MediaWiki adaptation to support Organizational Engineering* by Jorge Capela using Scalable vector graphics.

There are still some changes that would need to be done for the full cycle to be available, most of them related with Semantic MediaWiki, where the templates in most cases are different since we found some properties names to be ambiguous and connectors were not considered as unique elements with properties, but as parts of the shapes instead.

5. Conclusion

During our implementation phase we experienced many setbacks mentioned along the way that lead us to our final solution using a combination of Visio 2010 as the modeling tool, Visual Basic for Applications as the way to extract the diagram information from Visio and PHP parser to covert the outcome information from the diagrams into a XML file importable by Semantic MediaWiki where user friendly pages, with all the relevant information, were created.

One of major contributions of this thesis is the Universal Visio Metal Model for DEMO (UVMMD), where the base to gather all the important aspects in a DEMO diagram while working with Visio is set, allowing it to be applied to any change or new diagram that may come into use in the future in this methodology that is still taking its first steps.

The second major contribution was the expandable solution for the turning of the diagrams in SMW pages. Although not perfect, as it still requires specific diagram oriented details in the VBA phase, the PHP parser with the two easy to transform and expand XML files for the diagram properties and SMW templates is a very customizable option that allows great versatility for future developments.

Overall we believe we achieved our goal as we demonstrated with a pratical example that it is possible to integrate a modeling tool (Visio) with Semantic Media Wiki to allow the implementation of the DEMO methodology.

Bibliography

[1] J. Dietz, “Enterprise Ontology: Theory and Methodology”, Germany: Springer-Verlag Berlin Heidelberg, 2006.

[2] “Ontology”, http://en.wikipedia.org/wiki/Ontology, 2012.

[3] D. Aveiro, Engenharia Organizacional lectures, Universidade da Madeira, 2010.

[4] L. Apostel, “Towards the formal study of models in the non-formal sciences. Synthese 12”, 1960.

[5] J. Dietz, “Way of Working”, http://www.demo.nl/publications/doc_view/122-demo-3-way-of-working, 2009.

[6] J.Dietz, “A World Ontology Specification Language”, OTM 2005 Workshops, 2005.

[7] D. Aveiro, “G.O.D. (Generation, Operationalization & Discontinuation) and Control (sub)organizations: a DEMO-based approach for continuous real-time management of organizational change caused by exceptions,” Instituto Superior Técnico, 2010.

[8] “Microsoft Visio 2010 purchase information”, http://office.microsoft.com/en-us/visio/visio-2010-buy-page-FX101836377.aspx , 2011

[9] “Xemod - Licence and Price”, http://www.mprise.eu/default.aspx?id=106, 2011.

[10] “Purchase SmartDraw”, http://www.smartdraw.com/buy/, 2011

[11]“Dia”, http://live.gnome.org/Dia, 2011

[12] “Visual Paradigm for UML”,http://www.visual-paradigm.com/product/vpuml/, 2011

[13] “Enterprise Architect”, http://www.sparxsystems.com/products/ea/index.html, 2011

[14] “Kivio”, http://wiki.koffice.org/index.php?title=Kivio, 2011

[15] J. Capela, “Semantic MediaWiki adaptation to support Organizational Engineering”, Universidade da Madeira, 2011

[16] “MediaWiki”,http://en.wikipedia.org/wiki/MediaWiki, 2012

[17] “Size of Wikipedia”, http://en.wikipedia.org/wiki/Wikipedia:Size_of_Wikipedia, 2012

[18] “Semantic wiki”, http://en.wikipedia.org/wiki/Semantic_wiki, 2012

[19] “Semantic MediaWiki” http://en.wikipedia.org/wiki/Semantic_MediaWiki, 2012

[20] Introduction to Semantic MediaWiki, http://semantic-mediawiki.org/wiki/Help:Introduction_to_Semantic_MediaWiki, 2012

[21] Y. Wang, "*transformation of DEMO models into exchangeable format*", 2009

[22] "Demo Methodology", http://www.demo.nl/methodology, 2012

Appendix

A.1 Library Case

The following text is the complete transcription of the Library study case presented in J. Dietz, *Enterprise Ontology:*

"The library described hereafter is the small but autonomous public library of Delftown. In the building in which it is located, are a desk for lending books, called the out-desk, and a desk for returning books, called the indesk.

The in-desk is occupied by Sanne and the out-desk by Tim and Kris, in turn. There is a third desk, called the information desk, which is occupied by Lisa. At the information desk one can get information such as opening hours, loan rules, and membership fees. There is also a binder on Lisa's desk that contains the complete library catalog, sorted in several ways (by author, by category, and by title). One can freely browse through the binder to find the book one is looking for. Next to that, one can ask Lisa any question about the library, e.g., about the contents of the books in the catalog. The information desk also serves as the registration desk. Anyone who wants to be registered as member of the library has to apply with

Lisa. She writes the data needed on a registration form. The requested data are: surname, first name, middle initials, city of residence, street name, house number, postal code, sex, date of birth, starting date of the membership, and annual fee. By default, the annual fee is the standard annual fee as determined by the library board. Exceptions may be made for people without means. In that case, Lisa applies in writing to the library board for the reduced fee, which is a symbolic 1€ per year. The applicant then has to fill out a form in which a specification of the income and expenses in the past calendar year are asked for. This form is attached to Lisa's letter. The registration forms regarding regular memberships are collected daily (after closing time) by Sanne who puts the data in the Library Information System (LIS) that runs on the only PC of the library. LIS automatically prints a membership card and an invoice for every new member. The invoice is for the remaining months of the current calendar year, including the current month. So, for example, if one registers in September, one has to pay 4/12 of the annual fee. Both the card and the invoice can be collected by the regular new members from the next day on at the information desk. An applicant, then, also gets a letter of welcome, informing the new member about the library rules. Membership cards have a bar code on them representing the membership number. They are handed over to the new member after cash payment of the membership fee. For applicants of the reduced fee, the procedure is slightly different. They have to wait until they are informed in writing about the decision of the board. This is something Tim takes care of. As soon as he gets the decision of the board, he writes a corresponding note and sends it by postal mail to the applicant. A copy of the note goes to Lisa. If the reduced fee is allowed by the board, Lisa takes the registration form out of her drawer and hands it over to Sanne, for processing at the end of the day. In the case of a negative decision by the board, she inserts the form in the file of declined applications. Applicants whose

application has been declined are supposed to have canceled their original request for registration. They may, of course, register again, but only as fully paying members. The books that can be borrowed are put on shelves, and sorted by the category of the book (title). There may be several copies of the same book (title). Every such book copy is uniquely identified by a bar code. This code contains both the ISBN (International Standard Book Number) and the serial number of the book copy. If someone wants to borrow a book, he or she has to take (a copy of) the book from the shelves to the out-desk. Tim or Kris will then scan the bar code on the membership card, as well as the bar code on the book. These data are automatically entered into LIS. The book is now considered to be lent to the member. No more than five books may be lent simultaneously to the same member. When one returns a book, one goes to the in-desk and hands the book to Sanne. She scans the book code, which is automatically entered into LIS. On the screen of her computer, she sees whether the loan period is exceeded or not. If it is, she also sees the fine that has to be paid. The person who returns the book has to pay the fine right away and in cash.

After payment, Sanne marks the book in her computer as returned. If the loan period is not exceeded, she enters only that the book has been returned.

Returned books are piled on a table next to Sanne. About every hour, Lisa collects the pile and puts the books back on the shelves. While she is doing that, the information desk is temporarily unoccupied. Every month, the librarian decides which titles should be added and how many copies per title have to be ordered. She does so on the basis of the announcements of new books she knows of (by means of fliers of publishers, and also by surfing the Web) and on the basis of analysis reports of the reading habits of the members provided by LIS. The librarian exhausts an annual budget for buying new books that is decided upon by the board of the library. An order is directed to one publisher, but it may regard a number of copies of a number of book titles. At the start of a new calendar year, Kris sends out invoices to all current members for the annual membership fee. Fees have to be paid in cash the next time a member comes to the library to borrow a book. She also sends renewal requests of reduced fee memberships to the board. Attached to them are statements of income and expenses over the past year that she has asked the applicants to produce.

She deals with the decisions by the board in the same way as was done at the time of (the first) application."

A.2 **PHP** Code Kivio Parser

The following is the code created to extract the properties from the XML resultant from the Kivio exportation of a DEMO diagram.

```
<?php
echo "<pre>";

$file = "data.xml";
echo $file."\n";
global $inTag;

$inTag = "";
$xml_parser = xml_parser_create();
xml_parser_set_option($xml_parser, XML_OPTION_CASE_FOLDING, 0);
xml_parser_set_option($xml_parser, XML_OPTION_SKIP_WHITE, 1);
xml_set_processing_instruction_handler($xml_parser, "pi_handler");
xml_set_default_handler($xml_parser, "parseDEFAULT");
xml_set_element_handler($xml_parser, "startElement", "endElement");
xml_set_character_data_handler($xml_parser, "contents");

if (!($fp = fopen($file, "r"))) {
   if (!xml_parse($xml_parser, $data, feof($fp))) {
      die( sprintf("XML error: %s at line %d",
                  xml_error_string(xml_get_error_code($xml_parser)),
                  xml_get_current_line_number($xml_parser)));
   }
}
while ($data = fread($fp, 4096)) {
   if (!xml_parse($xml_parser, $data, feof($fp))) {
      die( sprintf("XML error: %s at line %d",
                  xml_error_string(xml_get_error_code($xml_parser)),
                  xml_get_current_line_number($xml_parser)));
   }
}
xml_parser_free($xml_parser);

function startElement($parser, $name, $attrs) {
   global $inTag;
   global $depth;
         $shapes[$value] = array();
   $padTag = str_repeat(str_pad(" ", 3), $depth);
         If (($name == "KivioSMLStencil" || $name == "KivioConnectorTarget" || $name == "KivioConnectorPoint"
|| $name == "KivioArrowHead" || $name == "KivioConnectorTargetList")){
                  foreach ($attrs as $key => $value) {
                        If ($key == "id" || $key == "targetId" || $key == "type"){
                           echo "\n$padTag$name";
                                 echo "\n$padTag".str_pad(" ", 3);
                                 echo " $key=\"$value\"";
                                       $shapes[$value] = $value;
                        }
                  }
         }else if ($name == "KivioConnectorList"){
                  echo "\n\n$padTag$name";
         }
$inTag = $name;
$depth++;
         foreach ($shapes as $ikey=>$value) {
                  if ($ikey != ""){
                        echo "<br/> ---> $ikey<br/>";
                  }
```

```
            }
}

function endElement($parser, $name) {
   global $depth;
            global $inTag;
   global $closeTag;
   $depth--;
  if ($closeTag == TRUE) {
     $inTag = "";
  } elseif ($inTag == $name) {
     $inTag = "";
  } else {
      $padTag = str_repeat(str_pad(" ", 3), $depth);
   }
}

function contents($parser, $data) {
   global $closeTag;
   $data = preg_replace("/^\s+/", "", $data);
   $data = preg_replace("/\s+$/", "", $data);
   if (!($data == ""))  {
      echo "&gt;$data";
      $closeTag = TRUE;
   } else {
      $closeTag = FALSE;
    }
}

function parseDEFAULT($parser, $data) {
   $data = preg_replace("/</", "&lt;", $data);
   $data = preg_replace("/>/", "&gt;", $data);
   echo $data;
}

function pi_handler($parser, $target, $data) {
   echo "&lt;?$target $data?&gt;\n";
}
echo "</pre>";
?>
```

A.3 SMW Installation guide

The following guide is a complete transcription from J. Capela's master thesis entitled *Semantic MediaWiki adaptation to support Organizational Engineering:*

"The OS used for this installation was Windows Vista, but the install steps should be the same for all Windows OS. To take advantage of the semantic features and correctly view the diagrams in SVG format the recommended browser is Opera (Firefox also works but doesn't show correctly SVG files). All files are included in the project CD.

Installation steps:

1. Install «xampp-win32-1.7.1-installer.exe» as Administrator

a) The installation dir should be «c:\», and the files will be automatically placed inside «c:\xampp\» folder

2. Open your browser and run phpMyAdmin

a) Create the user «wikiuser», password «12345» with all permissions

3. Extract « mediawiki-1.15.3.tar.gz» files to «htdocs» folder on xampp server

4. Rename the extracted folder to «wiki»

5. Change the «Só de leitura» property to allow writing in «wiki» folder (do the same to «config» folder)

6. Browse «http://localhost/wiki» to start the installation process

7. The form should be filled with the following information:

a) Wiki name: EOMediaWiki

b) Admin username: WikiSysop

c) Password: 12345

d) Database name: wikidb

e) DB username: wikiuser

f) DB password: 12345

*g) Database table prefix: eomw_*

h) Storage Engine: InnoDB

8. After completion of the installation process, copy «LocalSettings.php» from «config» folder to «wiki» folder root

9. Rename «config» folder (i.e. «old_config»)

10. Extract «smw-1.5.1_1.zip» files to wiki «extensions» folder

11.Browse «http://localhost/wiki» and login as Administrator:

a) Username: WikiSysop

b) Password: 12345

12. Browse «http://localhost/wiki/index.php/Special:SMWAdmin»

13. Click the «Initialise or upgrade tables» button

14. Click the «Start updating data» button

15. Browse «http://localhost/wiki/index.php/Special:SMWAdmin» and wait for the completion of the installation process (refresh the page)

16. Browse «http://localhost/wiki/index.php/Special:Version» and check if it was properly installed

17. Extract «scriptmanager-1.0.0_0.zip» files to wiki «extensions» folder

18. Extract «SemanticGraph-byJorgeCapela.zip» file to wiki «extensions» folder

19. Extract «SemanticDEMO-byJorgeCapela.zip» file to wiki «extensions» folder

20. Extract «semantic_forms_2.0.9.zip» file to wiki «extensions» folder

21. Extract the SMWHalo extension files from «smwhalo-1.5.1_1.zip» to wiki «extensions» folder

22. Extract the SMWHalo skin files from «smwhalo-1.5.1_1.zip» to wiki «skins» folder

23. Add the following lines at the end of «LocalSettings.php» file (in the exact same order)

```
$phpInterpreter=">C:\xampp\php\php.exe";

require_once("extensions/ScriptManager/SM_Initialize.php");

## Semantic MediaWiki
include_once("$IP/extensions/SemanticMediaWiki/SemanticMediaWiki.php");
enableSemantics('localhost:80');

include_once('extensions/SMWHalo/includes/SMW_Initialize.php');
enableSMWHalo(/*param-start-enableSMWHalo*/'SMWHaloStore2', NULL, NULL/*param-end-enableSMWHalo*/);

$wgDefaultSkin = 'ontoskin2'; // Set ontoskin2 as default for the entire wiki
$wgUseAjax = true; //MUST
$smwgServer="localhost:80";
$smwhgEnableLogging = false; // No logging
$smwgDeployVersion = true; // Deploy version is faster
$smwgAllowNewHelpQuestions = true;

## SemanticForms Extension
include_once("$IP/extensions/SemanticForms/SemanticForms.php");

## SemanticGraph Extension
require_once("$IP/extensions/SemanticGraph/includes/SemanticGraph2.php");

## SemanticDEMO Extension
require_once("$IP/extensions/SemanticDEMO/includes/SemanticDEMO.php");

## Runphp extension
require_once("$IP/extensions/runphp_page.php");
```

Image 67: «LocalSettings.php» file

24. Open a command-line interface and change dir to «C:\xampp\htdocs\wiki\extensions\SemanticMediaWiki\maintenance\»

25. Run the script «SMW_setup.php» to initialize the database tables: «C:\xampp\htdocs\wiki\extensions\SemanticMediaWiki\maintenance>C:\xampp\php\php.exe SMW_setup.php»

26. Extract «smwhalo-deploy-1.2.1_1.zip» file to wiki root folder

27. Add the GNU-Patch to the environment variable:

a) Right-click My Computer > Properties

b) Click Advanced tab

c) Click Environment variables

d) In the section System Variable select the variable Path

e) Click Edit

f) Add a semicolon ";" after the last value and then enter the path of the GNUPath: «c:\xampp\htdocs\wiki\deployment\tools»

g) Click OK and close the window

28. Switch to the command-line interface and navigate to the main directory of PHP: «C:\xampp\php»

29. Now start the patching process with the following command (all in the same command-line): «C:\xampp\php\php.exe C:\xampp\htdocs\wiki\deployment\tools\patch.php –d C:\xampp\htdocs\wiki –p C:\xampp\htdocs\wiki\extensions\SMWHalo\patch.txt»

30. After completion of the patching process, activate the Autocomplete feature:

a) Browse «http://localhost/wiki»

b) If you are not logged in, login as WikiSysop

c) Go to preferences

d) On Skins tab, choose «ontoskin2» (this option should appear selected, as was defined in the «LocalSettings.php» file)

e) On Misc tab, choose «Auto-triggered auto-completion»

f) Save changes

Testing the Installation

1. Browse «http://localhost/wiki/index.php/Special:Version» and you should see SMW+ Extension (version nn) listed as a Parser Hook

2. Now browse ««http://localhost/wiki» and:

a) Create a regular wiki page named "TestSMW" and in it enter the wiki text: Property test: [[testproperty::Dummypage]] [[Category:Test]]

b) Create the page "Category:Test" and insert some dummy text

3. Browse «http://localhost/wiki/index.php/Special:OntologyBrowser»

a) As soon as the interface is loaded, you should see "Test" in the category tree

b) Clicking on it and you should see "TestSMW" in the instance view

c) Clicking further on it, should display "testproperty Dummypage" in the properties view

A.4 XML Pattern for Semantic MediaWiki Exportation

The following is the full content of the XML file containing the pattern of templates needed for our Semantic MediaWiki exportation (ATD and OFD):

```
<SMWImport>
	<DiagramType>
		<DiagramKind>ATD</DiagramKind>
		<ShapeType>
			<ShapeKind>actor</ShapeKind>
			<PropertyList>
				<PropertyName>"\n|Represented in
Diagram=CurrentDiagram:</PropertyName>
				<PropertyName>"\n|Actor ID=Number:</PropertyName>
				<PropertyName>"\n|Actor Name=Name:</PropertyName>
				<PropertyName>"\n|Actor Description=Description:</PropertyName>
				<PropertyName>"\n|Actor Type=Type:</PropertyName>
				<PropertyName>"\n|Shape type=Visio Name:</PropertyName>
				<PropertyName>"\n|Connectors X=ConnectorX:</PropertyName>
				<PropertyName>"\n|Connectors Y=ConnectorY:</PropertyName>
				<PropertyName>"\n|Width=Width:</PropertyName>
				<PropertyName>"\n|Height=Height:</PropertyName>
				<PropertyName>"\n|Position X=Position X:</PropertyName>
				<PropertyName>"\n|Position Y=Position Y:</PropertyName>
			</PropertyList>
		</ShapeType>
		<ShapeKind>Transaction</ShapeKind>
			<PropertyList>
				<PropertyName>"\n|Represented in
Diagram=CurrentDiagram:</PropertyName>
				<PropertyName>"\n|Transaction ID=Number:</PropertyName>
				<PropertyName>"\n|Transaction Name=Name:</PropertyName>
				<PropertyName>"\n|Transaction Description=Description:</PropertyName>
				<PropertyName>"\n|Shape type=Visio Name:</PropertyName>
				<PropertyName>"\n|Connectors X=</PropertyName>
				<PropertyName>"\n|Connectors Y=</PropertyName>
				<PropertyName>"\n|Width=Width:</PropertyName>
				<PropertyName>"\n|Height=Height:</PropertyName>
				<PropertyName>"\n|Position X=Position X:</PropertyName>
				<PropertyName>"\n|Position Y=Position Y:</PropertyName>
			</PropertyList>
		</ShapeType>
		<ConnectorType>
			<ConnectorKind>Initiator</ConnectorKind>
			<PropertyList>
				<PropertyName>"\n|Represented in
Diagram=CurrentDiagram:</PropertyName>
				<PropertyName>"\n|Executing Actor Role=RelatedShape1:</PropertyName>
				<PropertyName>"\n|Executing Actor Role
CP=RelatedShape1CP:</PropertyName>
				<PropertyName>"\n|Executed
Transaction=RelatedShape2:</PropertyName>
				<PropertyName>"\n|Executed Transaction
CP=RelatedShape2CP:</PropertyName>
				<PropertyName>"\n|Shape Type=Visio Name:</PropertyName>
```

```
                                    <PropertyName>"\n|Starting Point
X=RelatedShape1CPPointX:</PropertyName>
                                    <PropertyName>"\n|Starting Point
Y=RelatedShape1CPPointY:</PropertyName>
                                    <PropertyName>"\n|End Point X=RelatedShape2CPPointX:</PropertyName>
                                    <PropertyName>"\n|End Point Y=RelatedShape2CPPointY:</PropertyName>
                                    <PropertyName>"\n|Moves X=MoveX:</PropertyName>
                                    <PropertyName>"\n|Moves Y=MoveY:</PropertyName>
                            </PropertyList>
                    </ConnectorType>
                    <ConnectorType>
                            <ConnectorKind>Executor</ConnectorKind>
                            <PropertyList>
                                    <PropertyName>"\n|Represented in
Diagram=CurrentDiagram:</PropertyName>
                                    <PropertyName>"\n|Executing Actor Role=RelatedShape1:</PropertyName>
                                    <PropertyName>"\n|Executing Actor Role
CP=RelatedShape1CP:</PropertyName>
                                    <PropertyName>"\n|Executed
Transaction=RelatedShape2:</PropertyName>
                                    <PropertyName>"\n|Executed Transaction
CP=RelatedShape2CP:</PropertyName>
                                    <PropertyName>"\n|Shape Type=Visio Name:</PropertyName>
                                    <PropertyName>"\n|Starting Point
X=RelatedShape1CPPointX:</PropertyName>
                                    <PropertyName>"\n|Starting Point
Y=RelatedShape1CPPointY:</PropertyName>
                                    <PropertyName>"\n|End Point X=RelatedShape2CPPointX:</PropertyName>
                                    <PropertyName>"\n|End Point Y=RelatedShape2CPPointY:</PropertyName>
                                    <PropertyName>"\n|Moves X=MoveX:</PropertyName>
                                    <PropertyName>"\n|Moves Y=MoveY:</PropertyName>
                            </PropertyList>
                    </ConnectorType>
            </DiagramType>
            <DiagramType>
                    <DiagramKind>OFD</DiagramKind>
                    <ShapeType>
                            <ShapeKind>facttype</ShapeKind>
                            <PropertyList>
                                    <PropertyName>"\n|Represented in
Diagram=CurrentDiagram:</PropertyName>
                                    <PropertyName>"\n|Encapsulated
Roles=EncapsulatedRoles:</PropertyName>
                                    <PropertyName>"\n|Fact Roles=FactRole:</PropertyName>
                                    <PropertyName>"\n|Fact Formulation=FactFormulation:</PropertyName>
                                    <PropertyName>"\n|fact type Type=Type:</PropertyName>
                                    <PropertyName>"\n|Description=|Description:</PropertyName>
                                    <PropertyName>"\n|Shape Type=Visio Name:</PropertyName>
                                    <PropertyName>"\n|Connectors X=</PropertyName>
                                    <PropertyName>"\n|Connectors Y=</PropertyName>
                                    <PropertyName>"\n|Width=Width:</PropertyName>
                                    <PropertyName>"\n|Height=Height:</PropertyName>
                                    <PropertyName>"\n|Position X=Position X:</PropertyName>
                                    <PropertyName>"\n|Position Y=Position Y:</PropertyName>
                            </PropertyList>
```

```
		</ShapeType>
		<ShapeType>
			<ShapeKind>resulttype</ShapeKind>
			<PropertyList>
				<PropertyName>"\n|Represented in
Diagram=CurrentDiagram:</PropertyName>
				<PropertyName>"\n|Encapsulated
Roles=EncapsulatedRoles:</PropertyName>
				<PropertyName>"\n|Result Type ID=Number:</PropertyName>
				<PropertyName>"\n|Result Formulation=Formulation:</PropertyName>
				<PropertyName>"\n|Variable =Variable:</PropertyName>
				<PropertyName>"\n|Result Type Type=Type:</PropertyName>
				<PropertyName>"\n|Description=|Description:</PropertyName>
				<PropertyName>"\n|Shape Type=Visio Name:</PropertyName>
				<PropertyName>"\n|Connectors X=</PropertyName>
				<PropertyName>"\n|Connectors Y=</PropertyName>
				<PropertyName>"\n|Width=Width:</PropertyName>
				<PropertyName>"\n|Height=Height:</PropertyName>
				<PropertyName>"\n|Position X=Position X:</PropertyName>
				<PropertyName>"\n|Position Y=Position Y:</PropertyName>
			</PropertyList>
		</ShapeType>
		<ShapeType>
			<ShapeKind>objectclass</ShapeKind>
			<PropertyList>
				<PropertyName>"\n|Represented in
Diagram=CurrentDiagram:</PropertyName>
				<PropertyName>"\n|Object Class Name=Name:</PropertyName>
				<PropertyName>"\n|Object Class Type=Type:</PropertyName>
				<PropertyName>"\n|Description=|Description:</PropertyName>
				<PropertyName>"\n|Shape Type=Visio Name:</PropertyName>
				<PropertyName>"\n|Connectors X=</PropertyName>
				<PropertyName>"\n|Connectors Y=</PropertyName>
				<PropertyName>"\n|Width=Width:</PropertyName>
				<PropertyName>"\n|Height=Height:</PropertyName>
				<PropertyName>"\n|Position X=Position X:</PropertyName>
				<PropertyName>"\n|Position Y=Position Y:</PropertyName>
			</PropertyList>
		</ShapeType>
		<ShapeType>
			<ShapeKind>Scale</ShapeKind>
			<PropertyList>
				<PropertyName>"\n|Represented in
Diagram=CurrentDiagram:</PropertyName>
				<PropertyName>"\n|Scale Name=Name:</PropertyName>
				<PropertyName>"\n|Scale Type=Type:</PropertyName>
				<PropertyName>"\n|Description=|Description:</PropertyName>
				<PropertyName>"\n|Shape Type=Visio Name:</PropertyName>
				<PropertyName>"\n|Connectors X=</PropertyName>
				<PropertyName>"\n|Connectors Y=</PropertyName>
				<PropertyName>"\n|Width=Width:</PropertyName>
				<PropertyName>"\n|Height=Height:</PropertyName>
				<PropertyName>"\n|Position X=Position X:</PropertyName>
				<PropertyName>"\n|Position Y=Position Y:</PropertyName>
			</PropertyList>
```

```
		</ShapeType>
		<ShapeType>
			<ShapeKind>Extension</ShapeKind>
			<PropertyList>
				<PropertyName>"\n|Represented in
Diagram=CurrentDiagram:</PropertyName>
				<PropertyName>"\n|Extension Name=Name:</PropertyName>
				<PropertyName>"\n|Encapsulated
Roles=EncapsulatedRoles:</PropertyName>
				<PropertyName>"\n|Variable =Variable:</PropertyName>
				<PropertyName>"\n|Description=|Description:</PropertyName>
				<PropertyName>"\n|Shape Type=Visio Name:</PropertyName>
				<PropertyName>"\n|Connectors X=</PropertyName>
				<PropertyName>"\n|Connectors Y=</PropertyName>
				<PropertyName>"\n|Width=Width:</PropertyName>
				<PropertyName>"\n|Height=Height:</PropertyName>
				<PropertyName>"\n|Position X=Position X:</PropertyName>
				<PropertyName>"\n|Position Y=Position Y:</PropertyName>
			</PropertyList>
		</ShapeType>
		<ConnectorType>
			<ConnectorKind>Unicity Law</ConnectorKind>
			<PropertyList>
				<PropertyName>"\n|Represented in
Diagram=CurrentDiagram:</PropertyName>
				<PropertyName>"\n|Shape=RelatedShape1:</PropertyName>
				<PropertyName>"\n|Beginning Shape
CP=RelatedShape1CP:</PropertyName>
				<PropertyName>"\n|End Shape CP=RelatedShape2CP:</PropertyName>
				<PropertyName>"\n|Shape Type=Visio Name:</PropertyName>
				<PropertyName>"\n|Starting Point
X=RelatedShape1CPPointX:</PropertyName>
				<PropertyName>"\n|Starting Point
Y=RelatedShape1CPPointY:</PropertyName>
				<PropertyName>"\n|End Point X=RelatedShape2CPPointX:</PropertyName>
				<PropertyName>"\n|End Point Y=RelatedShape2CPPointY:</PropertyName>
				<PropertyName>"\n|Moves X=MoveX:</PropertyName>
				<PropertyName>"\n|Moves Y=MoveY:</PropertyName>
			</PropertyList>
		</ConnectorType>
		<ConnectorType>
			<ConnectorKind>Reference Law</ConnectorKind>
			<PropertyList>
				<PropertyName>"\n|Represented in
Diagram=CurrentDiagram:</PropertyName>
				<PropertyName>"\n|Beginning Shape=RelatedShape1:</PropertyName>
				<PropertyName>"\n|Beginning Shape
CP=RelatedShape1CP:</PropertyName>
				<PropertyName>"\n|End Shape=RelatedShape2:</PropertyName>
				<PropertyName>"\n|End Shape CP=RelatedShape2CP:</PropertyName>
				<PropertyName>"\n|Shape Type=Visio Name:</PropertyName>
				<PropertyName>"\n|Starting Point
X=RelatedShape1CPPointX:</PropertyName>
				<PropertyName>"\n|Starting Point
Y=RelatedShape1CPPointY:</PropertyName>
```

```
                                        <PropertyName>"\n|End Point X=RelatedShape2CPPointX:</PropertyName>
                                        <PropertyName>"\n|End Point Y=RelatedShape2CPPointY:</PropertyName>
                                        <PropertyName>"\n|Moves X=MoveX:</PropertyName>
                                        <PropertyName>"\n|Moves Y=MoveY:</PropertyName>
                                </PropertyList>
                        </ConnectorType>
                        <ConnectorType>
                                <ConnectorKind>dependency law</ConnectorKind>
                                <PropertyList>
                                        <PropertyName>"\n|Represented in
Diagram=CurrentDiagram:</PropertyName>
                                        <PropertyName>"\n|Beginning Shape=RelatedShape1:</PropertyName>
                                        <PropertyName>"\n|Beginning Shape
CP=RelatedShape1CP:</PropertyName>
                                        <PropertyName>"\n|End Shape=RelatedShape2:</PropertyName>
                                        <PropertyName>"\n|End Shape CP=RelatedShape2CP:</PropertyName>
                                        <PropertyName>"\n|Shape Type=Visio Name:</PropertyName>
                                        <PropertyName>"\n|Starting Point
X=RelatedShape1CPPointX:</PropertyName>
                                        <PropertyName>"\n|Starting Point
Y=RelatedShape1CPPointY:</PropertyName>
                                        <PropertyName>"\n|End Point X=RelatedShape2CPPointX:</PropertyName>
                                        <PropertyName>"\n|End Point Y=RelatedShape2CPPointY:</PropertyName>
                                        <PropertyName>"\n|Moves X=MoveX:</PropertyName>
                                        <PropertyName>"\n|Moves Y=MoveY:</PropertyName>
                                </PropertyList>
                        </ConnectorType>
                </DiagramType>
</SMWImport>
```

A.5 XML Property List for each Shape

The following is the full content of the XML file containing the properties list for each shape (ATD, PSD and OFD):

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<DEMO>

	<DiagramType>
		<DiagramName>ATD</DiagramName>

		<ShapeType>
			<ShapeName>boundary</ShapeName>
			<UnaryPropertyType>
				<PropertyName>Type:</PropertyName>
				<PropertyName>Name:</PropertyName>
				<PropertyName>WidthShape:</PropertyName>
				<PropertyName>HeightShape:</PropertyName>
				<PropertyName>Position X:</PropertyName>
				<PropertyName>Position Y:</PropertyName>
			</UnaryPropertyType>
			<MultiplePropertyType>
				<PropertyName>ConnectorX</PropertyName>
				<PropertyName>ConnectorY</PropertyName>
			</MultiplePropertyType>
		</ShapeType>

		<ShapeType>
			<ShapeName>actor</ShapeName>
			<UnaryPropertyType>
				<PropertyName>Type:</PropertyName>
				<PropertyName>Name:</PropertyName>
				<PropertyName>Number:</PropertyName>
				<PropertyName>Position X:</PropertyName>
				<PropertyName>Position Y:</PropertyName>
				<PropertyName>WidthShape:</PropertyName>
				<PropertyName>HeightShape:</PropertyName>
			</UnaryPropertyType>
			<MultiplePropertyType>
				<PropertyName>ConnectorX</PropertyName>
				<PropertyName>ConnectorY</PropertyName>
			</MultiplePropertyType>
		</ShapeType>

		<ShapeType>
			<ShapeName>transaction</ShapeName>
			<UnaryPropertyType>
				<PropertyName>Type:</PropertyName>
				<PropertyName>Name:</PropertyName>
				<PropertyName>Number:</PropertyName>
				<PropertyName>Position X:</PropertyName>
				<PropertyName>Position Y:</PropertyName>
				<PropertyName>WidthShape:</PropertyName>
				<PropertyName>HeightShape:</PropertyName>
```

```
                </UnaryPropertyType>
                <MultiplePropertyType>
                        <PropertyName>ConnectorX</PropertyName>
                        <PropertyName>ConnectorY</PropertyName>
                </MultiplePropertyType>
        </ShapeType>

        <ConnectorType>
                <ConnectorName>initiator</ConnectorName>
                <UnaryPropertyType>
                        <PropertyName>Type:</PropertyName>
                        <PropertyName>Begining Shape:</PropertyName>
                        <PropertyName>Begining Shape CP:</PropertyName>
                        <PropertyName>End Shape:</PropertyName>
                        <PropertyName>End Shape CP:</PropertyName>
                        <PropertyName>Position X:</PropertyName>
                        <PropertyName>Position Y:</PropertyName>
                </UnaryPropertyType>
                <MultiplePropertyType>
                        <PropertyName>MoveX</PropertyName>
                        <PropertyName>MoveY</PropertyName>
                </MultiplePropertyType>
        </ConnectorType>

        <ConnectorType>
                <ConnectorName>executor</ConnectorName>
                <UnaryPropertyType>
                        <PropertyName>Type:</PropertyName>
                        <PropertyName>Begining Shape:</PropertyName>
                        <PropertyName>Begining Shape CP:</PropertyName>
                        <PropertyName>End Shape:</PropertyName>
                        <PropertyName>End Shape CP:</PropertyName>
                        <PropertyName>Position X:</PropertyName>
                        <PropertyName>Position Y:</PropertyName>
                </UnaryPropertyType>
                <MultiplePropertyType>
                        <PropertyName>MoveX</PropertyName>
                        <PropertyName>MoveY</PropertyName>
                </MultiplePropertyType>
        </ConnectorType>
</DiagramType>

<DiagramType>
        <DiagramName>OFD</DiagramName>

        <ShapeType>
                <ShapeName>facttype</ShapeName>
                <UnaryPropertyType>
                        <PropertyName>Type:</PropertyName>
                        <PropertyName>FactFormulation:</PropertyName>
                        <PropertyName>Variable:</PropertyName>
                        <PropertyName>EncapsulatedRoles:</PropertyName>
                        <PropertyName>Number:</PropertyName>
                        <PropertyName>Position X:</PropertyName>
                        <PropertyName>Position Y:</PropertyName>
```

```
                <PropertyName>WidthShape:</PropertyName>
                <PropertyName>HeightShape:</PropertyName>
        </UnaryPropertyType>
        <MultiplePropertyType>
                <PropertyName>FactRoleN</PropertyName>
                <PropertyName>MoveX</PropertyName>
                <PropertyName>MoveY</PropertyName>
                <PropertyName>ConnectorX</PropertyName>
                <PropertyName>ConnectorY</PropertyName>
        </MultiplePropertyType>
</ShapeType>
<ShapeType>
        <ShapeName>resulttype</ShapeName>
        <UnaryPropertyType>
                <PropertyName>Type:</PropertyName>
                <PropertyName>Formulation:</PropertyName>
                <PropertyName>Variable:</PropertyName>
                <PropertyName>EncapsulatedRoles:</PropertyName>
                <PropertyName>Number:</PropertyName>
                <PropertyName>Position X:</PropertyName>
                <PropertyName>Position Y:</PropertyName>
                <PropertyName>WidthShape:</PropertyName>
                <PropertyName>HeightShape:</PropertyName>
        </UnaryPropertyType>
        <MultiplePropertyType>
                <PropertyName>MoveX</PropertyName>
                <PropertyName>MoveY</PropertyName>
                <PropertyName>ConnectorX</PropertyName>
                <PropertyName>ConnectorY</PropertyName>
        </MultiplePropertyType>
</ShapeType>
<ShapeType>
        <ShapeName>objectclass</ShapeName>
        <UnaryPropertyType>
                <PropertyName>Type:</PropertyName>
                <PropertyName>Name:</PropertyName>
                <PropertyName>TypeFormat:</PropertyName>
                <PropertyName>Position X:</PropertyName>
                <PropertyName>Position Y:</PropertyName>
                <PropertyName>WidthShape:</PropertyName>
                <PropertyName>HeightShape:</PropertyName>
        </UnaryPropertyType>
        <MultiplePropertyType>
                <PropertyName>MoveX</PropertyName>
                <PropertyName>MoveY</PropertyName>
                <PropertyName>ConnectorX</PropertyName>
                <PropertyName>ConnectorY</PropertyName>
        </MultiplePropertyType>
</ShapeType>
<ShapeType>
        <ShapeName>extension</ShapeName>
        <UnaryPropertyType>
                <PropertyName>Type:</PropertyName>
                <PropertyName>Name:</PropertyName>
                <PropertyName>Position X:</PropertyName>
```

```
                <PropertyName>Position Y:</PropertyName>
                <PropertyName>WidthShape:</PropertyName>
                <PropertyName>HeightShape:</PropertyName>
        </UnaryPropertyType>
        <MultiplePropertyType>
                <PropertyName>MoveX</PropertyName>
                <PropertyName>MoveY</PropertyName>
                <PropertyName>ConnectorX</PropertyName>
                <PropertyName>ConnectorY</PropertyName>
        </MultiplePropertyType>
</ShapeType>
<ShapeType>
        <ShapeName>scale</ShapeName>
        <UnaryPropertyType>
                <PropertyName>Type:</PropertyName>
                <PropertyName>Name:</PropertyName>
                <PropertyName>TypeFormat:</PropertyName>
                <PropertyName>Position X:</PropertyName>
                <PropertyName>Position Y:</PropertyName>
                <PropertyName>WidthShape:</PropertyName>
                <PropertyName>HeightShape:</PropertyName>
        </UnaryPropertyType>
        <MultiplePropertyType>
                <PropertyName>MoveX</PropertyName>
                <PropertyName>MoveY</PropertyName>
                <PropertyName>ConnectorX</PropertyName>
                <PropertyName>ConnectorY</PropertyName>
        </MultiplePropertyType>
</ShapeType>

<ConnectorType>
        <ConnectorName>unicitylaw</ConnectorName>
        <UnaryPropertyType>
                <PropertyName>Type:</PropertyName>
                <PropertyName>Begining X:</PropertyName>
                <PropertyName>End X:</PropertyName>
                <PropertyName>Position X:</PropertyName>
                <PropertyName>Position Y:</PropertyName>
        </UnaryPropertyType>
        <MultiplePropertyType>
                <PropertyName>MoveX</PropertyName>
                <PropertyName>MoveY</PropertyName>
        </MultiplePropertyType>
</ConnectorType>
<ConnectorType>
        <ConnectorName>referencelaw</ConnectorName>
        <UnaryPropertyType>
                <PropertyName>Type:</PropertyName>
                <PropertyName>Begining X:</PropertyName>
                <PropertyName>End X:</PropertyName>
                <PropertyName>Position X:</PropertyName>
                <PropertyName>Position Y:</PropertyName>
        </UnaryPropertyType>
        <MultiplePropertyType>
                <PropertyName>MoveX</PropertyName>
```

```
                        <PropertyName>MoveY</PropertyName>
                </MultiplePropertyType>
        </ConnectorType>
        <ConnectorType>
                <ConnectorName>dependencylaw</ConnectorName>
                <UnaryPropertyType>
                        <PropertyName>Type:</PropertyName>
                        <PropertyName>Begining X:</PropertyName>
                        <PropertyName>End X:</PropertyName>
                        <PropertyName>Position X:</PropertyName>
                        <PropertyName>Position Y:</PropertyName>
                </UnaryPropertyType>
                <MultiplePropertyType>
                        <PropertyName>MoveX</PropertyName>
                        <PropertyName>MoveY</PropertyName>
                </MultiplePropertyType>
        </ConnectorType>
</DiagramType>

<DiagramType>
        <DiagramName>PSD</DiagramName>

        <ShapeType>
                <ShapeName>process</ShapeName>
                <UnaryPropertyType>
                        <PropertyName>Type:</PropertyName>
                        <PropertyName>Name:</PropertyName>
                        <PropertyName>Number:</PropertyName>
                        <PropertyName>Position X:</PropertyName>
                        <PropertyName>Position Y:</PropertyName>
                </UnaryPropertyType>
                <MultiplePropertyType>
                        <PropertyName>MoveX</PropertyName>
                        <PropertyName>MoveY</PropertyName>
                        <PropertyName>ConnectorX</PropertyName>
                        <PropertyName>ConnectorY</PropertyName>
                </MultiplePropertyType>
        </ShapeType>
        <ShapeType>
                <ShapeName>start</ShapeName>
                <UnaryPropertyType>
                        <PropertyName>Type:</PropertyName>
                        <PropertyName>Position X:</PropertyName>
                        <PropertyName>Position Y:</PropertyName>
                </UnaryPropertyType>
                <MultiplePropertyType>
                        <PropertyName>MoveX</PropertyName>
                        <PropertyName>MoveY</PropertyName>
                        <PropertyName>ConnectorX</PropertyName>
                        <PropertyName>ConnectorY</PropertyName>
                </MultiplePropertyType>
        </ShapeType>
        <ShapeType>
                <ShapeName>execution</ShapeName>
                <UnaryPropertyType>
```

```
                <PropertyName>Type:</PropertyName>
                <PropertyName>Position X:</PropertyName>
                <PropertyName>Position Y:</PropertyName>
        </UnaryPropertyType>
        <MultiplePropertyType>
                <PropertyName>MoveX</PropertyName>
                <PropertyName>MoveY</PropertyName>
                <PropertyName>ConnectorX</PropertyName>
                <PropertyName>ConnectorY</PropertyName>
        </MultiplePropertyType>
</ShapeType>
<ShapeType>
        <ShapeName>allow</ShapeName>
        <UnaryPropertyType>
                <PropertyName>Type:</PropertyName>
                <PropertyName>Position X:</PropertyName>
                <PropertyName>Position Y:</PropertyName>
        </UnaryPropertyType>
        <MultiplePropertyType>
                <PropertyName>MoveX</PropertyName>
                <PropertyName>MoveY</PropertyName>
                <PropertyName>ConnectorX</PropertyName>
                <PropertyName>ConnectorY</PropertyName>
        </MultiplePropertyType>
</ShapeType>
<ShapeType>
        <ShapeName>accept</ShapeName>
        <UnaryPropertyType>
                <PropertyName>Type:</PropertyName>
                <PropertyName>Position X:</PropertyName>
                <PropertyName>Position Y:</PropertyName>
        </UnaryPropertyType>
        <MultiplePropertyType>
                <PropertyName>MoveX</PropertyName>
                <PropertyName>MoveY</PropertyName>
                <PropertyName>ConnectorX</PropertyName>
                <PropertyName>ConnectorY</PropertyName>
        </MultiplePropertyType>
</ShapeType>
<ShapeType>
        <ShapeName>cancel(ac)</ShapeName>
        <UnaryPropertyType>
                <PropertyName>Type:</PropertyName>
                <PropertyName>Position X:</PropertyName>
                <PropertyName>Position Y:</PropertyName>
        </UnaryPropertyType>
        <MultiplePropertyType>
                <PropertyName>MoveX</PropertyName>
                <PropertyName>MoveY</PropertyName>
                <PropertyName>ConnectorX</PropertyName>
                <PropertyName>ConnectorY</PropertyName>
        </MultiplePropertyType>
</ShapeType>
<ShapeType>
        <ShapeName>cancel(rq)</ShapeName>
```

```
		<UnaryPropertyType>
			<PropertyName>Type:</PropertyName>
			<PropertyName>Position X:</PropertyName>
			<PropertyName>Position Y:</PropertyName>
		</UnaryPropertyType>
		<MultiplePropertyType>
			<PropertyName>MoveX</PropertyName>
			<PropertyName>MoveY</PropertyName>
			<PropertyName>ConnectorX</PropertyName>
			<PropertyName>ConnectorY</PropertyName>
		</MultiplePropertyType>
</ShapeType>
<ShapeType>
		<ShapeName>cancel(pm)</ShapeName>
		<UnaryPropertyType>
			<PropertyName>Type:</PropertyName>
			<PropertyName>Position X:</PropertyName>
			<PropertyName>Position Y:</PropertyName>
		</UnaryPropertyType>
		<MultiplePropertyType>
			<PropertyName>MoveX</PropertyName>
			<PropertyName>MoveY</PropertyName>
			<PropertyName>ConnectorX</PropertyName>
			<PropertyName>ConnectorY</PropertyName>
		</MultiplePropertyType>
</ShapeType>
<ShapeType>
		<ShapeName>cancel(st)</ShapeName>
		<UnaryPropertyType>
			<PropertyName>Type:</PropertyName>
			<PropertyName>Position X:</PropertyName>
			<PropertyName>Position Y:</PropertyName>
		</UnaryPropertyType>
		<MultiplePropertyType>
			<PropertyName>MoveX</PropertyName>
			<PropertyName>MoveY</PropertyName>
			<PropertyName>ConnectorX</PropertyName>
			<PropertyName>ConnectorY</PropertyName>
		</MultiplePropertyType>
</ShapeType>
<ShapeType>
		<ShapeName>Decline</ShapeName>
		<UnaryPropertyType>
			<PropertyName>Type:</PropertyName>
			<PropertyName>Position X:</PropertyName>
			<PropertyName>Position Y:</PropertyName>
		</UnaryPropertyType>
		<MultiplePropertyType>
			<PropertyName>MoveX</PropertyName>
			<PropertyName>MoveY</PropertyName>
			<PropertyName>ConnectorX</PropertyName>
			<PropertyName>ConnectorY</PropertyName>
		</MultiplePropertyType>
</ShapeType>
<ShapeType>
```

```
		<ShapeName>Promise</ShapeName>
		<UnaryPropertyType>
			<PropertyName>Type:</PropertyName>
			<PropertyName>Position X:</PropertyName>
			<PropertyName>Position Y:</PropertyName>
		</UnaryPropertyType>
		<MultiplePropertyType>
			<PropertyName>MoveX</PropertyName>
			<PropertyName>MoveY</PropertyName>
			<PropertyName>ConnectorX</PropertyName>
			<PropertyName>ConnectorY</PropertyName>
		</MultiplePropertyType>
</ShapeType>
<ShapeType>
		<ShapeName>Quit</ShapeName>
		<UnaryPropertyType>
			<PropertyName>Type:</PropertyName>
			<PropertyName>Position X:</PropertyName>
			<PropertyName>Position Y:</PropertyName>
		</UnaryPropertyType>
		<MultiplePropertyType>
			<PropertyName>MoveX</PropertyName>
			<PropertyName>MoveY</PropertyName>
			<PropertyName>ConnectorX</PropertyName>
			<PropertyName>ConnectorY</PropertyName>
		</MultiplePropertyType>
</ShapeType>
<ShapeType>
		<ShapeName>Refuse</ShapeName>
		<UnaryPropertyType>
			<PropertyName>Type:</PropertyName>
			<PropertyName>Position X:</PropertyName>
			<PropertyName>Position Y:</PropertyName>
		</UnaryPropertyType>
		<MultiplePropertyType>
			<PropertyName>MoveX</PropertyName>
			<PropertyName>MoveY</PropertyName>
			<PropertyName>ConnectorX</PropertyName>
			<PropertyName>ConnectorY</PropertyName>
		</MultiplePropertyType>
</ShapeType>
<ShapeType>
		<ShapeName>Reject</ShapeName>
		<UnaryPropertyType>
			<PropertyName>Type:</PropertyName>
			<PropertyName>Position X:</PropertyName>
			<PropertyName>Position Y:</PropertyName>
		</UnaryPropertyType>
		<MultiplePropertyType>
			<PropertyName>MoveX</PropertyName>
			<PropertyName>MoveY</PropertyName>
			<PropertyName>ConnectorX</PropertyName>
			<PropertyName>ConnectorY</PropertyName>
		</MultiplePropertyType>
</ShapeType>
```

```
<ShapeType>
	<ShapeName>Request</ShapeName>
	<UnaryPropertyType>
		<PropertyName>Type:</PropertyName>
		<PropertyName>Position X:</PropertyName>
		<PropertyName>Position Y:</PropertyName>
	</UnaryPropertyType>
	<MultiplePropertyType>
		<PropertyName>MoveX</PropertyName>
		<PropertyName>MoveY</PropertyName>
		<PropertyName>ConnectorX</PropertyName>
		<PropertyName>ConnectorY</PropertyName>
	</MultiplePropertyType>
</ShapeType>
<ShapeType>
	<ShapeName>state</ShapeName>
	<UnaryPropertyType>
		<PropertyName>Type:</PropertyName>
		<PropertyName>Position X:</PropertyName>
		<PropertyName>Position Y:</PropertyName>
	</UnaryPropertyType>
	<MultiplePropertyType>
		<PropertyName>MoveX</PropertyName>
		<PropertyName>MoveY</PropertyName>
		<PropertyName>ConnectorX</PropertyName>
		<PropertyName>ConnectorY</PropertyName>
	</MultiplePropertyType>
</ShapeType>
<ShapeType>
	<ShapeName>stop</ShapeName>
	<UnaryPropertyType>
		<PropertyName>Type:</PropertyName>
		<PropertyName>Position X:</PropertyName>
		<PropertyName>Position Y:</PropertyName>
	</UnaryPropertyType>
	<MultiplePropertyType>
		<PropertyName>MoveX</PropertyName>
		<PropertyName>MoveY</PropertyName>
		<PropertyName>ConnectorX</PropertyName>
		<PropertyName>ConnectorY</PropertyName>
	</MultiplePropertyType>
</ShapeType>

<ConnectorType>
	<ConnectorName>connection</ConnectorName>
	<UnaryPropertyType>
		<PropertyName>Type:</PropertyName>
		<PropertyName>Begining X:</PropertyName>
		<PropertyName>End X:</PropertyName>
		<PropertyName>Position X:</PropertyName>
		<PropertyName>Position Y:</PropertyName>
	</UnaryPropertyType>
	<MultiplePropertyType>
		<PropertyName>MoveX</PropertyName>
		<PropertyName>MoveY</PropertyName>
```

```
                </MultiplePropertyType>
            </ConnectorType>
            <ConnectorType>
                <ConnectorName>waitconnection</ConnectorName>
                <UnaryPropertyType>
                    <PropertyName>Type:</PropertyName>
                    <PropertyName>Begining X:</PropertyName>
                    <PropertyName>End X:</PropertyName>
                    <PropertyName>Position X:</PropertyName>
                    <PropertyName>Position Y:</PropertyName>
                </UnaryPropertyType>
                <MultiplePropertyType>
                    <PropertyName>MoveX</PropertyName>
                    <PropertyName>MoveY</PropertyName>
                </MultiplePropertyType>
            </ConnectorType>
        </DiagramType>

</DEMO>
```

A.6 VBA Code Visio (ATD and OFD)

The following is the VBA code needed to extract all the properties considered as needed from a ATD or OFD diagram represented in Visio.

```
Sub teste2()
   Dim pagObj As Visio.Page
   Dim shpsObj As Visio.Shapes
   Dim shpObj As Visio.Shape
   Dim celObj As Visio.Cell
   Dim OrderInfo() As String
   Dim Moves() As String
   Dim iShapeCount As Integer
   Dim i As Integer
   Dim j As Integer
   Dim n As Integer
   Dim filename As String
   Dim teste As String
   Dim aux As String
   Dim aux2 As String
   Dim aux3 As Double
   Dim aux4 As Double
   Dim aux5 As Double
   Dim realwidth As Double
   Dim aux6 As String
   Dim aux7 As String
   Dim aux8 As String
   Dim Length As Integer
   Open "c:\OFDInfoV2.txt" For Output As #1 ' Abrir Documento
   Set pagObj = ActivePage
   Set shpsObj = pagObj.Shapes
   iShapeCount = shpsObj.Count
   ReDim OrderInfo(222, iShapeCount - 1)
   For i = 1 To iShapeCount
      Set shpObj = shpsObj(i)
      'recolhe o nome da forma (tipo e numero de instancia no diagrama)
      OrderInfo(0, i - 1) = shpObj.Name
      'recolhe o valor da propriedade Name
      If shpObj.CellExists("Prop.Name", visExistsLocally) Then
         Set celObj = shpObj.Cells("Prop.Name")
         OrderInfo(1, i - 1) = celObj.ResultStr("")
      End If
      'recolhe o valor da propriedade Number
      If shpObj.CellExists("Prop.Number", visExistsLocally) Then
         Set celObj = shpObj.Cells("Prop.Number")
         OrderInfo(2, i - 1) = celObj.ResultStr("")
      End If
      'recolhe a forma de origem e o conector ao qual está ligado
      If shpObj.CellExists("BeginX", visExistsLocally) Then
         Set celObj = shpObj.Cells("BeginX")
         OrderInfo(3, i - 1) = celObj.Formula
      End If
      'recolhe a forma de destino e qual o conector a que se vai ligar
      If shpObj.CellExists("EndX", visExistsLocally) Then
         Set celObj = shpObj.Cells("EndX")
```

```
    OrderInfo(4, i - 1) = celObj.Formula
End If
'recolhe a posição no eixo x do diagrama da forma
If shpObj.CellExists("PinX", visExistsLocally) Then
    Set celObj = shpObj.Cells("PinX")
    OrderInfo(5, i - 1) = celObj.Formula
End If
'recolhe a posição no eixo y do diagrama da forma
If shpObj.CellExists("PinY", visExistsLocally) Then
    Set celObj = shpObj.Cells("PinY")
    OrderInfo(6, i - 1) = celObj.Formula
End If
'recolhe a largura da forma
If shpObj.CellExists("Width", visExistsLocally) Then
    Set celObj = shpObj.Cells("Width")
    OrderInfo(7, i - 1) = celObj.Formula
End If
'recolhe a altura da forma
If shpObj.CellExists("Height", visExistsLocally) Then
    Set celObj = shpObj.Cells("Height")
    OrderInfo(8, i - 1) = celObj.Formula
End If
'recolhe o valor da propriedade Cardinalidade
If shpObj.CellExists("n-aryProp.N", visExistsLocally) Then
    Set celObj = shpObj.Cells("n-aryProp.N")
    OrderInfo(9, i - 1) = celObj.Formula
End If
'recolhe o valor da propriedade Formulação
If shpObj.CellExists("Prop.Formulation", visExistsLocally) Then
    Set celObj = shpObj.Cells("Prop.Formulation")
    OrderInfo(10, i - 1) = celObj.ResultStr("")
End If
'recolhe o valor da propriedade Type Format
If shpObj.CellExists("Prop.Type", visExistsLocally) Then
    Set celObj = shpObj.Cells("Prop.Type")
    OrderInfo(11, i - 1) = celObj.ResultStr("")
End If
'recolhe o valor da propriedade VariableName1
If shpObj.CellExists("Prop.VariableName1", visExistsLocally) Then
    Set celObj = shpObj.Cells("Prop.VariableName1")
    OrderInfo(12, i - 1) = celObj.ResultStr("")
End If
'recolhe o valor da propriedade VariableName2
If shpObj.CellExists("Prop.VariableName2", visExistsLocally) Then
    Set celObj = shpObj.Cells("Prop.VariableName2")
    OrderInfo(13, i - 1) = celObj.ResultStr("")
End If
'recolhe o valor da propriedade FactRoleName1
If shpObj.CellExists("Prop.FactRoleName1", visExistsLocally) Then
    Set celObj = shpObj.Cells("Prop.FactRoleName1")
    OrderInfo(14, i - 1) = celObj.ResultStr("")
End If
'recolhe o valor da propriedade FactRoleName2
If shpObj.CellExists("Prop.FactRoleName2", visExistsLocally) Then
    Set celObj = shpObj.Cells("Prop.FactRoleName2")
```

```
   OrderInfo(15, i - 1) = celObj.ResultStr("")
End If
'recolhe o valor da propriedade FactRoleName3
If shpObj.CellExists("Prop.FactRoleName3", visExistsLocally) Then
   Set celObj = shpObj.Cells("Prop.FactRoleName3")
   OrderInfo(16, i - 1) = celObj.ResultStr("")
End If
'recolhe o valor da propriedade FactRoleName4
If shpObj.CellExists("Prop.FactRoleName4", visExistsLocally) Then
   Set celObj = shpObj.Cells("Prop.FactRoleName4")
   OrderInfo(17, i - 1) = celObj.ResultStr("")
End If
'recolhe o valor da propriedade FactRoleName5
If shpObj.CellExists("Prop.FactRoleName5", visExistsLocally) Then
   Set celObj = shpObj.Cells("Prop.FactRoleName5")
   OrderInfo(18, i - 1) = celObj.ResultStr("")
End If
'recolhe o valor da propriedade FactRoleName6
If shpObj.CellExists("Prop.FactRoleName6", visExistsLocally) Then
   Set celObj = shpObj.Cells("Prop.FactRoleName6")
   OrderInfo(19, i - 1) = celObj.ResultStr("")
End If
'recolhe o valor da propriedade N
If shpObj.CellExists("Prop.N", visExistsLocally) Then
   Set celObj = shpObj.Cells("Prop.N")
   OrderInfo(20, i - 1) = celObj.Formula
End If
'recolhe a formulação de um fact type
If shpObj.CellExists("Prop.FactFormulation", visExistsLocally) Then
   Set celObj = shpObj.Cells("Prop.FactFormulation")
   OrderInfo(21, i - 1) = celObj.Formula
End If
'ciclo para os connectores
j = 50
n = 1
Do While j < 150
'Posição no eixo X
   If shpObj.CellExists("Connections.X" & n, visExistsLocally) Then
      Set celObj = shpObj.Cells("Connections.X" & n)
      OrderInfo(j, i - 1) = celObj.Formula
   End If

'Posição no eixo Y

   j = j + 1
   If shpObj.CellExists("Connections.Y" & n, visExistsLocally) Then
      Set celObj = shpObj.Cells("Connections.Y" & n)
      OrderInfo(j, i - 1) = celObj.Formula
   End If
   n = n + 1
   j = j + 1
Loop
'ciclo para os movimentos
j = 150
n = 1
```

```
    Do While j < 200
    'Movimentos no eixo X
      If shpObj.CellExists("Geometry1.X" & n, visExistsLocally) Then
        Set celObj = shpObj.Cells("Geometry1.X" & n)
        OrderInfo(j, i - 1) = celObj.Formula
      End If
    'Movimentos no eixo Y
      j = j + 1
      If shpObj.CellExists("Geometry1.Y" & n, visExistsLocally) Then
        Set celObj = shpObj.Cells("Geometry1.Y" & n)
        OrderInfo(j, i - 1) = celObj.Formula
      End If
      n = n + 1
      j = j + 1
    Loop
    If shpObj.CellExists("BegTrigger", visExistsLocally) Then
      Set celObj = shpObj.Cells("BegTrigger")
      OrderInfo(221, i - 1) = celObj.Formula
    End If
    If shpObj.CellExists("EndTrigger", visExistsLocally) Then
      Set celObj = shpObj.Cells("EndTrigger")
      OrderInfo(222, i - 1) = celObj.Formula
    End If


    Set shpObj = Nothing
  Next i
  For i = 0 To pagObj.Shapes.Count - 1
      Print #1, "Element"; i  ' Elemento
      If OrderInfo(1, i) <> "" Then
        Print #1, "Label: "; OrderInfo(1, i) ' Info
      End If
      If OrderInfo(21, i) <> "" Then
        aux = Trim(Replace(OrderInfo(21, i), """", ""))
        Print #1, "Label: "; OrderInfo(21, i) ' Info
      End If
      If OrderInfo(10, i) <> "" Then
        Print #1, "Label: "; OrderInfo(10, i) ' Info
      End If
      If OrderInfo(0, i) <> "" Then
        Print #1, "Visio Name: "; OrderInfo(0, i) ' Info
      End If
      If OrderInfo(1, i) <> "" Then
        Print #1, "Name: "; OrderInfo(1, i) ' Info
      End If
      If OrderInfo(2, i) <> "" Then
        Print #1, "Number: "; OrderInfo(2, i) ' Info
      End If
      If OrderInfo(221, i) <> "" Then
        aux = Trim(Replace(OrderInfo(221, i), "_XFTRIGGER(", ""))
        aux = Trim(Replace(aux, "!EventXFMod)", ""))
        If InStr(OrderInfo(221, i), "transaction") >= 1 Then Print #1, "End Shape: "; aux ' Info
        If InStr(OrderInfo(221, i), "actor") >= 1 Then Print #1, "Begining Shape: "; aux ' Info
        If InStr(OrderInfo(221, i), "fact") >= 1 Then Print #1, "Begining Shape: "; aux ' Info
        If InStr(OrderInfo(221, i), "result") >= 1 Then Print #1, "Begining Shape: "; aux ' Info
```

```
   If InStr(OrderInfo(221, i), "scale") >= 1 Then Print #1, "Begining Shape: "; aux ' Info
   If InStr(OrderInfo(221, i), "object") >= 1 Then Print #1, "End Shape: "; aux ' Info
   If InStr(OrderInfo(221, i), "extension") >= 1 Then Print #1, "End Shape: "; aux ' Info
End If
If OrderInfo(3, i) <> "" Then
   aux6 = "unicity law"
   If InStr(OrderInfo(0, i), aux6) = 1 Then
      aux6 = Trim(Replace(OrderInfo(3, i), "PAR(PNT(", ""))
      aux6 = Trim(Replace(aux6, "!onnections.X", ""))
      aux6 = Trim(Replace(aux6, "!onnections.Y", ""))
      aux6 = Trim(Replace(aux6, "))", ""))
      aux6 = Trim(Replace(aux6, ";", ""))
      aux6 = Mid(aux6, 1, 13)
      aux6 = Trim(Replace(aux6, "!", ""))
      aux6 = Trim(Replace(aux6, "C", ""))
      Print #1, "Begining Shape: "; aux6 ' Info
      aux7 = Trim(Replace(OrderInfo(3, i), "PAR(PNT(", ""))
      aux7 = Trim(Replace(aux7, "!Connections.", ""))
      aux7 = Trim(Replace(aux7, "!Connections.", ""))
      aux7 = Trim(Replace(aux7, "))", ""))
      aux7 = Trim(Replace(aux7, "fact type", ""))
      Length = Len(aux7)
      aux7 = Mid(aux7, (Length - 1), 3)
      aux7 = Trim(Replace(aux7, "Y", ""))
      Print #1, "Begining Shape CP: "; aux7 ' Info
   End If
   If InStr(OrderInfo(0, i), "unicity law") = 0 Then
      aux2 = Trim(Replace(OrderInfo(3, i), "PAR(PNT(", ""))
      aux2 = Trim(Replace(aux2, aux, ""))
      aux2 = Trim(Replace(aux2, "!Connections.X", ""))
      aux2 = Trim(Replace(aux2, "!Connections.Y", ""))
      aux2 = Trim(Replace(aux2, "))", ""))
      aux2 = Trim(Replace(aux2, ";", ""))
      aux2 = Mid(aux2, 1, 1)
      If InStr(OrderInfo(221, i), "transaction") >= 1 Then Print #1, "End Shape CP: "; aux2 ' Info
      If InStr(OrderInfo(221, i), "actor") >= 1 Then Print #1, "Begining Shape CP: "; aux2 ' Info
   If InStr(OrderInfo(221, i), "fact") >= 1 Then Print #1, "Begining Shape CP: "; aux2 ' Info
   If InStr(OrderInfo(221, i), "result") >= 1 Then Print #1, "Begining Shape CP: "; aux2 ' Info
   If InStr(OrderInfo(221, i), "scale") >= 1 Then Print #1, "Begining Shape CP: "; aux2 ' Info
   If InStr(OrderInfo(221, i), "object") >= 1 Then Print #1, "End Shape CP: "; aux2 ' Info
   If InStr(OrderInfo(221, i), "extension") >= 1 Then Print #1, "End Shape CP: "; aux2 ' Info
    End If
End If
If OrderInfo(222, i) <> "" Then
   aux = Trim(Replace(OrderInfo(222, i), "_XFTRIGGER(", ""))
   aux = Trim(Replace(aux, "!EventXFMod)", ""))
   If InStr(OrderInfo(221, i), "transaction") >= 1 Then Print #1, "Begining Shape: "; aux ' Info
   If InStr(OrderInfo(221, i), "actor") >= 1 Then Print #1, "End Shape: "; aux ' Info
   If InStr(OrderInfo(221, i), "fact") >= 1 Then Print #1, "End Shape: "; aux ' Info
   If InStr(OrderInfo(221, i), "result") >= 1 Then Print #1, "End Shape: "; aux ' Info
   If InStr(OrderInfo(221, i), "scale") >= 1 Then Print #1, "End Shape: "; aux ' Info
   If InStr(OrderInfo(221, i), "object") >= 1 Then Print #1, "Begining Shape: "; aux ' Info
   If InStr(OrderInfo(221, i), "extension") >= 1 Then Print #1, "Begining Shape: "; aux ' Info
End If
If OrderInfo(4, i) <> "" Then
```

```
    aux6 = "unicity law"
    If InStr(OrderInfo(0, i), aux6) = 1 Then
       aux6 = Trim(Replace(OrderInfo(4, i), "PAR(PNT(", ""))
       aux6 = Trim(Replace(aux6, "!onnections.X", ""))
       aux6 = Trim(Replace(aux6, "!onnections.Y", ""))
       aux6 = Trim(Replace(aux6, "))", ""))
       aux6 = Trim(Replace(aux6, ";", ""))
       aux6 = Mid(aux6, 1, 13)
       aux6 = Trim(Replace(aux6, "!", ""))
       aux6 = Trim(Replace(aux6, "C", ""))
       Print #1, "End Shape: "; aux6 ' Info
       aux7 = Trim(Replace(OrderInfo(4, i), "PAR(PNT(", ""))
       aux7 = Trim(Replace(aux7, "!Connections.", ""))
       aux7 = Trim(Replace(aux7, "!Connections.", ""))
       aux7 = Trim(Replace(aux7, "))", ""))
       aux7 = Trim(Replace(aux7, "fact type", ""))
       Length = Len(aux7)
       aux7 = Mid(aux7, (Length - 1), 3)
       aux7 = Trim(Replace(aux7, "Y", ""))
       Print #1, "End Shape CP: "; aux7 ' Info
    End If
    If InStr(OrderInfo(0, i), "unicity law") = 0 Then
       aux2 = Trim(Replace(OrderInfo(4, i), "PAR(PNT(", ""))
       aux2 = Trim(Replace(aux2, aux, ""))
       aux2 = Trim(Replace(aux2, "!Connections.X", ""))
       aux2 = Trim(Replace(aux2, "!Connections.Y", ""))
       aux2 = Trim(Replace(aux2, "))", ""))
       aux2 = Trim(Replace(aux2, ";", ""))
       aux2 = Mid(aux2, 1, 1)
       If InStr(OrderInfo(221, i), "transaction") >= 1 Then Print #1, "Begining Shape CP: "; aux2 ' Info
       If InStr(OrderInfo(221, i), "actor") >= 1 Then Print #1, "End Shape CP: "; aux2 ' Info
       If InStr(OrderInfo(221, i), "fact") >= 1 Then Print #1, "End Shape CP: "; aux2 ' Info
       If InStr(OrderInfo(221, i), "result") >= 1 Then Print #1, "End Shape CP: "; aux2 ' Info
       If InStr(OrderInfo(221, i), "scale") >= 1 Then Print #1, "End Shape CP: "; aux2 ' Info
       If InStr(OrderInfo(221, i), "object") >= 1 Then Print #1, "Begining Shape CP: "; aux2 ' Info
       If InStr(OrderInfo(221, i), "extension") >= 1 Then Print #1, "Begining Shape CP: "; aux2 ' Info
    End If
End If
If OrderInfo(5, i) <> "" Then
       aux6 = "extension."
       If InStr(OrderInfo(0, i), aux6) = 1 Then
          aux6 = Trim(Replace(OrderInfo(5, i), "PNTX(LOCTOPAR(PNT(", ""))
          aux6 = Trim(Replace(aux6, "onnections.X2;", ""))
          aux6 = Trim(Replace(aux6, "onnections.Y2);", ""))
          aux6 = Trim(Replace(aux6, "ventXFMod;EventXFMod))+0", ""))
          aux6 = Mid(aux6, 1, 13)
          aux6 = Trim(Replace(aux6, "C", ""))
          aux6 = Trim(Replace(aux6, "!", ""))
          aux7 = Trim(Replace(OrderInfo(5, i), "PNTX(LOCTOPAR(PNT(", ""))
          aux7 = Trim(Replace(aux7, "Connections.", ""))
          aux7 = Trim(Replace(aux7, "EventXFMod;EventXFMod))+0", ""))
          aux7 = Trim(Replace(aux7, "fact type", ""))
          aux7 = Trim(Replace(aux7, "! mm", ""))
          Length = Len(aux7)
          aux7 = Mid(aux7, (Length - 7), 3)
```

```
            aux7 = Trim(Replace(aux7, "Y", ""))
            aux7 = Trim(Replace(aux7, "!", ""))
            aux7 = Trim(Replace(aux7, ")", ""))
            Print #1, "Starting X: "; aux6 ' Info
            Print #1, "Starting CP: "; aux7 ' Info
        End If
        If InStr(OrderInfo(0, i), "extension.") = 0 Then
            aux = Trim(Replace(OrderInfo(5, i), "mm", ""))
            If IsNumeric(Trim(aux)) Then Print #1, "Position X: "; aux ' Info
        End If
End If
If OrderInfo(6, i) <> "" Then
    aux6 = "extension."
    If InStr(OrderInfo(0, i), aux6) = 1 Then
        aux6 = Trim(Replace(OrderInfo(6, i), "PNTY(LOCTOPAR(PNT(", ""))
        aux6 = Trim(Replace(aux6, "onnections.X2;", ""))
        aux6 = Trim(Replace(aux6, "onnections.Y2);", ""))
        aux6 = Trim(Replace(aux6, "ventXFMod;EventXFMod))+0", ""))
        aux6 = Mid(aux6, 1, 13)
        aux6 = Trim(Replace(aux6, "C", ""))
        aux6 = Trim(Replace(aux6, "!", ""))
        aux7 = Trim(Replace(OrderInfo(6, i), "PNTY(LOCTOPAR(PNT(", ""))
        aux7 = Trim(Replace(aux7, "Connections.", ""))
        aux7 = Trim(Replace(aux7, "EventXFMod;EventXFMod))+0", ""))
        aux7 = Trim(Replace(aux7, "fact type", ""))
        aux7 = Trim(Replace(aux7, "! mm", ""))
        Length = Len(aux7)
        aux7 = Mid(aux7, (Length - 7), 3)
        aux7 = Trim(Replace(aux7, "Y", ""))
        aux7 = Trim(Replace(aux7, "!", ""))
        aux7 = Trim(Replace(aux7, ")", ""))
        Print #1, "Starting Y: "; aux6 ' Info
        Print #1, "Starting CP: "; aux7 ' Info
    End If
    If InStr(OrderInfo(0, i), "extension.") = 0 Then
        aux = Trim(Replace(OrderInfo(6, i), "mm", ""))
        If IsNumeric(Trim(aux)) Then Print #1, "Position Y: "; aux ' Info
    End If
End If
If OrderInfo(20, i) <> "" Then
    aux = Trim(Replace(OrderInfo(20, i), """", ""))
    If IsNumeric(Trim(aux)) Then Print #1, "EncapsulatedRoles: "; aux ' Info
End If
If OrderInfo(7, i) <> "" Then
     aux = Trim(Replace(OrderInfo(7, i), "mm", ""))
     If IsNumeric(Trim(aux)) Then realwidth = CDbl(aux)
     If IsNumeric(Trim(aux)) Then Print #1, "Width: "; realwidth
     If Not IsNumeric(Trim(aux)) Then
         aux = Trim(Replace(OrderInfo(7, i), "cm*Prop.N", ""))
         If IsNumeric(Trim(aux)) Then aux3 = CDbl(aux)
         aux2 = Trim(Replace(OrderInfo(20, i), """", ""))
         If IsNumeric(Trim(aux2)) Then aux4 = CDbl(aux2)
         aux5 = (aux3 * 10) * aux4
         realwidth = aux5
         Print #1, "Width: "; realwidth
```

```
        End If
    '    aux = Trim(Replace(OrderInfo(7, i), "mm", ""))
    '    Print #1, "WidthShape: "; aux ' Info
    End If
    If OrderInfo(8, i) <> "" Then
       aux = Trim(Replace(OrderInfo(8, i), "mm", ""))
       If IsNumeric(Trim(aux)) Then Print #1, "Height: "; aux ' Info
    End If
    If OrderInfo(9, i) <> "" Then
       Print #1, "Cardinality: "; OrderInfo(9, i) ' Info
    End If
    If OrderInfo(10, i) <> "" Then
       Print #1, "Formulation: "; OrderInfo(10, i) ' Info
    End If
    If OrderInfo(11, i) <> "" Then
       Print #1, "TypeFormat: "; OrderInfo(11, i) ' Info
    End If
    If OrderInfo(12, i) <> "" Then
       Print #1, "Variable 1: "; OrderInfo(12, i) ' Info
    End If
    If OrderInfo(13, i) <> "" Then
       Print #1, "Variable 2: "; OrderInfo(13, i) ' Info
    End If
    If OrderInfo(14, i) <> "" Then
      If Not IsNumeric(Trim(OrderInfo(14, i))) Then Print #1, "FactRole 1: "; OrderInfo(14, i) ' Info
    End If
    If OrderInfo(15, i) <> "" Then
       If Not IsNumeric(Trim(OrderInfo(15, i))) Then Print #1, "FactRole 2: "; OrderInfo(15, i) ' Info
    End If
    If OrderInfo(16, i) <> "" Then
       If Not IsNumeric(Trim(OrderInfo(16, i))) Then Print #1, "FactRole 3: "; OrderInfo(16, i) ' Info
    End If
    If OrderInfo(17, i) <> "" Then
       If Not IsNumeric(Trim(OrderInfo(17, i))) Then Print #1, "FactRole 4: "; OrderInfo(17, i) ' Info
    End If
    If OrderInfo(18, i) <> "" Then
       If Not IsNumeric(Trim(OrderInfo(18, i))) Then Print #1, "FactRole 5: "; OrderInfo(18, i) ' Info
    End If
    If OrderInfo(19, i) <> "" Then
       If Not IsNumeric(Trim(OrderInfo(19, i))) Then Print #1, "FactRole 6: "; OrderInfo(19, i) ' Info
    End If
    If OrderInfo(21, i) <> "" Then
       aux = Trim(Replace(OrderInfo(21, i), """", ""))
       Print #1, "FactFormulation: "; OrderInfo(21, i) ' Info
    End If
  k = 1
  l = 0
  Do While k < 100
    l = l + 1
    If OrderInfo(49 + k, i) <> "" Then
       aux = Trim(Replace(OrderInfo(49 + k, i), "Width*", ""))
       aux2 = Trim(Replace(realwidth, "mm", ""))
       If IsNumeric(Trim(aux)) Then aux3 = CDbl(aux)
       If IsNumeric(Trim(aux2)) Then aux4 = CDbl(aux2)
       aux5 = aux3 * aux4
```

```
            Print #1, "ConnectorX " & l; ": "; aux5 ' Posição no eixo X do connector "l"
        End If
        k = k + 1
        If OrderInfo(49 + k, i) <> "" Then
            aux = Trim(Replace(OrderInfo(49 + k, i), "Height*", ""))
            aux2 = Trim(Replace(OrderInfo(8, i), "mm", ""))
            If IsNumeric(Trim(aux)) Then aux3 = CDbl(aux)
            If IsNumeric(Trim(aux2)) Then aux4 = CDbl(aux2)
            aux5 = aux3 * aux4
            Print #1, "ConnectorY " & l; ": "; aux5 ' Posição no eixo Y do connector "l"
        End If
        k = k + 1
    Loop
    k = 1
    l = 0
    Do While k < 50
        l = l + 1
        If OrderInfo(149 + k, i) <> "" Then
            aux = Trim(Replace(OrderInfo(149 + k, i), "mm", ""))
            If IsNumeric(aux) Then Print #1, "MoveX " & l; ": "; aux ' Movimento "l" no eixo X do connector
        End If
        k = k + 1
        If OrderInfo(149 + k, i) <> "" Then
            aux = Trim(Replace(OrderInfo(149 + k, i), "mm", ""))
            If IsNumeric(aux) Then Print #1, "MoveY " & l; ": "; aux ' Movimento "l" no eixo Y do connector
        End If
        k = k + 1
    Loop
        Print #1, ' Linha em branco
  Next i
    Close #1 ' Fechar documento
End Sub
```