Semantic Media Wiki Adaptation to Support Organizational Engineering

Adaptação do Semantic Media Wiki para Suporte e Engenharia Organizacional

DISSERTAÇÃO DE MESTRADO

Jorge Manuel Reis Capela
MESTRADO EM ENGENHARIA INFORMÁTICA

UNIVERSIDADE da MADEIRA
A Nossa Universidade
www.uma.pt

abril | 2012

Semantic Media Wiki Adaptation to Support Organizational Engineering

Adaptação do Semantic Media Wiki para Suporte e Engenharia Organizacional

DISSERTAÇÃO DE MESTRADO

Jorge Manuel Reis Capela
MESTRADO EM ENGENHARIA INFORMÁTICA

ORIENTAÇÃO
David Sardinha Andrade de Aveiro

Adaptação do *Semantic MediaWiki* para suporte a Engenharia Organizacional

Semantic MediaWiki adaptation to support Organizational Engineering

Dissertação para a obtenção do Grau de Mestre em

Engenharia Informática

Universidade da Madeira

Júri

Presidente: Professor Doutor Eduardo Leopoldo Fermé

Vogal: Professor Doutor Ian Oakley

Vogal (Orientador): Professor Doutor David Sardinha Andrade de Aveiro

Abril de 2012

Resumo

As organizações possuem processos de gestão e de operacionalização que se tornam cada vez mais diversificados e complexos. Estes processos podem partilhar diversas características, por exemplo, são: realizados por pessoas; constrangidos por recursos limitados e planeados, executados e controlados. Na maior parte das organizações, a característica «planeamento, execução e controlo» não é eficazmente colocada em prática: as pessoas diversificam funções mas os processos continuam complexos e muitas vezes são conduzidos sem uma estrutura lógica. Tal deve-se a diversos factores: as pessoas não conhecem exactamente a missão da organização, nem os seus papéis de actores enquanto recursos humanos; as organizações não possuem mecanismos adequados para lidar com os processos organizacionais, o que resulta num forte impacto na sua competitividade. A chave para garantir essa competitividade passa por descrever, estandardizar e adaptar a forma de responder a determinados tipos de eventos de negócio e a sua interacção com fornecedores, parceiros de negócio, organizações concorrentes e clientes.

Neste contexto surge a modelação associada à Engenharia Organizacional com o objectivo de construir representações de factos organizacionais para criar e desenvolver uma consciência colectiva da realidade organizacional (self-awareness) nas organizações. A abordagem desta dissertação centra-se na utilização da metodologia *DEMO* (*Design and Engineering Methodology for Organizations*), criada e desenvolvida por Jan L. G. Dietz, com o intuito de auxiliar a referida modelação através de um wiki semântico (*Semantic MediaWiki*).

O contexto deste projecto insere-se na adaptação do *Semantic MediaWiki* para suporte à Engenharia Organizacional e permitir a modelação dos processos organizacionais, baseada na metodologia *DEMO*. A nossa investigação e implementação procuram: formalizar factos organizacionais nas páginas do *Semantic MediaWiki*; e adaptar o *Semantic MediaWiki* para gerar automaticamente diagramas *DEMO* com base nas páginas wiki que contêm a formalização dos factos organizacionais.

Palavras-chave: Engenharia Organizacional, *DEMO Methodology*, *Semantic MediaWiki*, *Graphviz*, automatic diagram generation, SVG diagram.

Abstract

Organizations have operation and managing processes that become increasingly diverse and complex. These processes may share several characteristics, for example: are made by people; are constrained by limited resources; are planned, executed and controlled. In most organizations, the «planning, implementation and control» characteristic is not effectively put into practice: people diversify functions but processes remain complex and are often conducted without a logical structure. This is due to several factors: people do not know exactly the mission of the organization or their roles as actors as human resources; organizations do not have adequate mechanisms to deal with organizational processes, resulting in a strong impact on their competitiveness. The key to ensure that competitiveness is to describe, standardize and adapt the way to respond to certain types of business events and their interaction with suppliers, business partners, competitors and customers organizations.

In this context, emerges the modeling associated with Organizational Engineering with the aim of building organizational representations of facts to create and develop a collective consciousness of organizational reality (self-awareness) in organizations. The approach of this dissertation focuses on the use of the *DEMO* methodology (*Design and Engineering Methodology for Organizations*), created and developed by Jan L. G. Dietz, in order to assist such modeling through a semantic wiki (*Semantic MediaWiki*).

The context of this project is the adaptation of *Semantic MediaWiki* for Organizational Engineering support and enable the modeling of organizational processes based on the methodology *DEMO*. Our research and implementation seek: formalizing organizational facts using *Semantic MediaWiki* pages; and adapt *Semantic MediaWiki* to automatically generate *DEMO* diagrams based on the wiki pages that contain the formalization of organizational facts.

Keywords: Organizational Engineering, *DEMO Methodology*, *Semantic MediaWiki*, *Graphviz*, automatic diagram generation, SVG diagram.

Table of Contents

Table of Figures

Table of Tables

1 Introduction

Our project main focus will be the adaptation of *Semantic MediaWiki* to support Organizational Engineering. In the end we should be able to store textual information and graphical information related to an organization. By textual information we mean using *Semantic MediaWiki* to store organizational artifacts through semantic properties and assigning them some value. By graphical information we mean using *Semantic MediaWiki* to generate and manipulate diagrams that are representations of that textual information.

In this chapter we will set the project motivation, define our objectives, describe the project and its context, and make an approach to chapters content.

1.1 Motivation

IT projects (approximately 75%) fail to meet the expectations of its users. One of the main causes is an insufficient or inadequate knowledge of organizational reality to be automated or supported by an information system (IS). The Organizational Engineering course emerged in the 90's and adds concepts and engineering methods applied to organization with the aim of understand and represent many facets of the same, as well as facilitate analysis and organizational change, regardless of the implementation of ISs. Semantic Wikis are understandable and intuitive tools that can be used by people with minimal computer knowledge. It is intended that the employees of an organization can use a semantic wiki with the aim of creating and developing a collective awareness of organizational reality. This tool enables the collection of distributed and coherent organizational knowledge in the form of elements and semantic relations in models aligned with the organizational reality, enabling the capture and monitoring of their progress as well as a more efficient and effective development of ISs that support such a reality. This organizational reality should also be translated into graphical symbols that are critical elements in diagrams, achieving a better understanding of organizational knowledge.

1.2 Objectives

Our main objectives are the analysis and development of a prototype based on *MediaWiki* and *Semantic MediaWiki* that allows the modeling of organizations, based on *DEMO* methodology (*Design and Engineering Methodology for Organizations*) created and developed by Jan L. G. Dietz.

Our main challenges are formalizing organizational facts using *Semantic MediaWiki* pages and defining semantic properties to store and link all the information needed to generate diagrams. Thus, there is the need to adapt *Semantic MediaWiki* to automatically generate *DEMO* diagrams supported by those wiki pages and semantic properties. In an earlier stage we should use *Graphviz* software and a modified version of *SemanticGraph* wiki extension to accomplish the diagrams generation. In a final stage we will be developing a new extension, that we will name *SemanticDEMO*. This new extension will be capable of generation *DEMO* diagrams in a more efficient and powerful way.

1.3 Project description and context

The project context is semantic web and how it can be used to help model organizations. All the project work was related with *MediaWiki*, *Semantic MediaWiki*, *Graphviz*, a modified version of *SemanticGraph* extension, and a new extension called *SemanticDEMO*. We started by defining a way of formalizing organizational facts using *Semantic MediaWiki* pages, where each page represents a fact. Also each page contains several semantic properties that define the fact. Subsequent to this achievement we have adapted *Semantic MediaWiki* to automatically generate *DEMO* diagrams. First we modified *SemanticGraph* extension to generate the diagrams using *Graphviz*. Then we created a new extension called *SemanticDEMO* based on *SemanticGraph* extension, in order to generate diagrams in a more efficient way.

1.4 Content

We intended to create a structure that follows a strict guideline based on theoretical

research and implementation of practical concepts. Chapter 2 – «Research content and problems definition» presents the theoretical basis and the software used as the starting point of our research. We also identify the problems and define our research strategy. Chapter 3 – «Related work» looks at previous projects related to the subject with the aim of help solving our problems. The solutions found to address the problems we raised are described in Chapter 4 – «Solutions and contributions». A more detailed and technical explanation about the elaborated work follows in Chapter 5 – «Implementation». Chapter 6 – «Future work» talks about research directions to follow next. In Chapter 7– «Conclusion» we draw the project resultant conclusions.

2 Research content and problem definition

The present chapter covers the fundamental theoretical concepts needed in order to understand our project. First, we introduce the concepts of Enterprise Ontology, ontology of a world, World Ontology Specification Language, *DEMO* Methodology and ontological meta-model. These issues are interrelated and allow us to understand the need to model the organizational facts. Following we bring in the concepts of Semantic Web and Semantic Wikis, allowing us to recognize the importance of using semantic to represent those organizational facts. Then we take a look at the software used as a starting point for our research and for what we developed. Following we address some considerations regarding the use of browsers, pointing out the best way to visualize SVG files. In the end of the chapter we identify the problems we intend to solve with this project and set the research strategy.

2.1 Enterprise Ontology

Enterprise ontology is a novel subject. If you have ever heard about ontology before, it is most probable that it was in the context of the World-Wide Web, particularly in the context of the Semantic Web. This topic will be covered later in this chapter.

2.1.1 Definition

A widely adopted definition of ontology is that an ontology is a formal, explicit specification of a shared conceptualization. It regards the conceptualization of (a part of) the world, so it is something in our mind.[1]

Managing an enterprise and getting services from it as a client or collaborating with it as partner in a network, is nowadays far more complicated than it was in the past. The problems in current enterprises, of any kind, are well investigated and well documented. The common denominator of these problems is complexity. The knowledge that one acquires at management or business schools does not suffice anymore. Even gifted entrepreneur can nowadays not succeed without a basic, systematic, and integral understanding of how enterprises work.

In order to really cope with the current and the future challenges, a conceptual model of the enterprise is needed that is coherent, comprehensive, consistent, and concise, and that only shows the essence of the operation of an enterprise model. The distinguished aspect models must constitute a logical a truly integral while. All relevant issues must be covered and be free from contradictions or irregularities. The aspect models must not contain any superfluous matters, that the whole is compact and succinct. Thus, this conceptual model shows only the essence of the enterprise and it abstracts from all realization and implementation issues. We shall call such a conceptual model an ontological model.

An ontology must be explicit and clear, there should be no room for misunderstandings. It is specified in a formal way and natural language is inappropriate for this task, because of its inherent ambiguity and impreciseness. But for now, we need to introduce some notions related to the world and its state.

2.1.2 The Ontology of a World

The notion of world is a very general one: there is a world of traveling by airplane, there is a world of educating students at a university, there is a world of repairing and maintaining cars, and so on. A state of such a world can simply be conceived as a set of elementary facts, such as the fact that a particular person or car or insurance policy exists, or that a particular person owns a particular car, and that a particular insurance policy is for a particular car. This can be considered as factual knowledge, by which we mean the knowledge about the states and the state transitions of a world. At any moment a world is in a particular state, which is simply defined as a set of objects; these objects are said to be current during the time that the state prevails. A state change is called a transition. The occurrence of a transition is called an event. Consequently, a transition can take place several times during the lifetime of a world; events however are unique: they take place only once. An event is caused by an act.

In order to understand what a state of a world is, it is necessary to distinguish between two kinds of objects: *stata* (singular: *statum*) and *facta* (singular: *factum*). A *statum* is something that is the case, has always been the case, and will always be the case; it is constant (i.e., “the author of book title T is A”). The existence of these objects is

timeless. *Stata* are subject to existence laws. These laws require or prohibit the coexistence of stata (in the same state of a world). For example, if the author of some book is “Ludwig Wittgenstein”, it cannot also be “John Irving”.[1]

Contrary to a *statum*, a *factum* is the result or the effect of an act (i.e., “book title T has been published”). The becoming existent of a *factum* is an event. Before the occurrence of the event, it did not exist; after the occurrence it does exist. *Facta* are subject to occurrence laws. These laws require or prohibit sequences of events in the course of time. For example, after the creation of the *factum* “loan L has been started”, the event “loan L has been ended” might occur, and, in between, several other *facta* may have been created, such as “the fine for loan L has been paid”.

We are now able to provide a precise definition of the ontology, or, more precisely, the ontological model of a world: «The ontological model of a world consists of the specification of its state space and its transition space».

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 *statum* types of which instances may exist in a state of the world. The existence laws determine the inclusion or exclusion of the coexistence of *stata*. By the transition space is understood the set of allowed or lawful sequences of transitions. It is specified by the transition base and the occurrence laws. The transition base is the set of *factum* types of which instances may occur in the world. Every such instance has a time stamp, which is the event time. The occurrence laws determine the order in which *facta* are required or allowed to occur.

We get now to a point where we must specify the conceptual model using the World Ontology Specification Language (WOSL).

2.1.3 WOSL (World Ontology Specification Language)

WOSL is a language for the specification of world ontologies. In this project context we will use it only for the specification of the state model. This language is based on the philosophies of Bunge and Wittgenstein. Wittgenstein’s proposition “The world is the totality of facts, not of things” can be reconciled with Bunge’s notion of world by taking

into account that everything in Bunge's world is a thing in a class, on the basis of its properties. So, a thing in Bunge's world is a (unary) fact in Wittgenstein's world. Likewise, a thing in Wittgenstein's world corresponds with the bare individual in Bunge's world.[1]

What we will express in WOSL are ontological models, consisting of concepts or predicates. Every individual predicate represents an individual fact in the world. Because of the similarity between the ontology of a world and the conceptual schema of a database, the author has adopted a graphical notation that is applied by one of the fact oriented conceptual modeling languages, namely ORM.

As stated before on section 2.1.2 , to understand what a state of a world is, it is necessary to distinguish between two kinds of objects: *stata* and *facta*. WOSL language has several graphical pictures to represent these *stata* and *facta*. Figure 1 shows some *statum* type declarations.

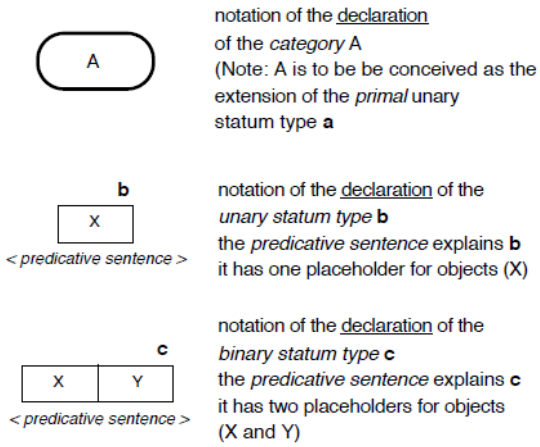

Figure 1: *Statum* type declarations

Figures 2 and 3 show the specification of existence laws.

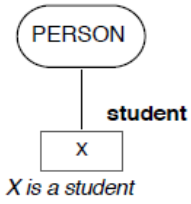

Figure 2: Example of a reference law

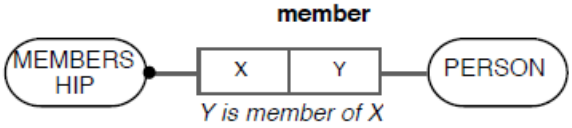

Figure 3: Example of a dependency law

Figure 4 shows an example of the declaration of a *factum* type.

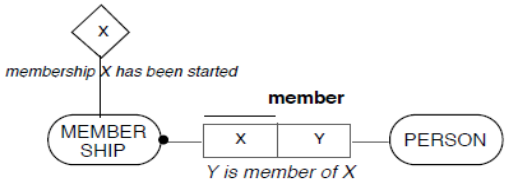

Figure 4: Example of a *factum* type

2.2 *DEMO* Methodology

DEMO is a methodology for the design, engineering, and implementation of organizations and networks of organizations. The entering into and complying with commitments is the operational principle for every organization. These commitments are established in the communication between social individuals, i.e. human beings.[2]

DEMO considers an organization to consist of a coherent layered integration of three aspect-organizations: the B-organization (business), the I-organization (information) and the D-organization (document). These constitute a coherent hierarchy, in which the I-organization supports the B-organization and the D-organization supports the I-organization. Every organizational change typically regards one of the aspect-organizations.

The four aspect models of *DEMO* (Construction Model, Process Model, State Model, and Action Model) are perspectives under the same meta-model. Any organizational

change has consequences in all aspect models.[1]

The *DEMO*-transaction is a universal pattern of coordination acts which lead to the creation of one new production fact. It is the generic building block for all business processes. One can use it as a template for the design of business processes with the assurance that no (relevant) action or information will be overseen. Actions like the promise and the acceptance (of organizational facts) are carried out tacitly and are rarely supported by information systems or workflow systems. Thus they are easily overseen in change projects. *DEMO* models are objective. Thus *DEMO* guarantees reproducible models, which are independent from the 'modelers'. These modelers are also independent from the momentary occupiers of the actor roles (the employees). Information needs no longer rely on what is said, but on what has been objectively established to be needed by an actor role.

As said before, the modeling of an enterprise can be achieved through the four aspect models that we distinguish, in which the ontological knowledge of an enterprise is expressed, such that this knowledge is easily accessible and manageable.

The Construction Model (CM) specifies the identified transaction types and the associated actor roles, as well as the information links between the actor roles and the information banks (the collective name for production banks and coordination banks); in short, the CM specifies the construction of the organization. A dashed line splits the CM triangle into two parts. The left part is the interaction model (IAM); it shows the active influences between actor roles: the execution of transactions. The right part is the interstriction model (ISM); it shows the passive influences between actor roles.

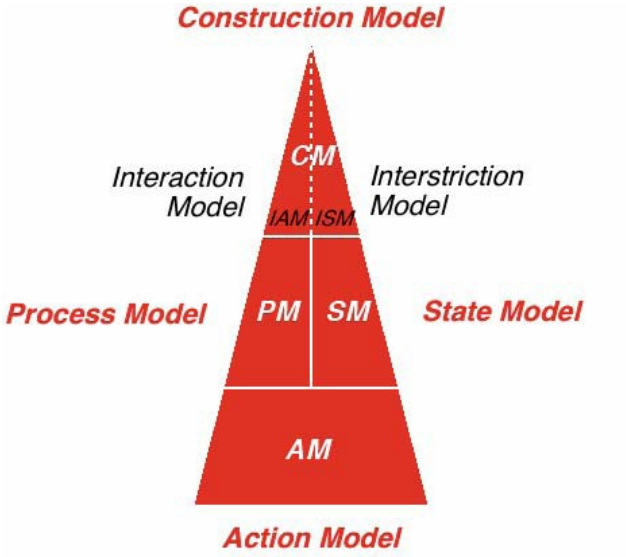

Figure 5: The ontological aspect models

The Process Model (PM) contains, for every transaction type in the CM, the specific transaction pattern of the transaction type. The PM also contains the causal and conditional relationships between transactions.

The Action Model (AM) specifies the action rules that serve as guidelines for the actors in dealing with their agenda. It contains one or more action rules for every agendum type. These rules are grouped according to the actor roles that are distinguished.

The State Model (SM) specifies the state space of the P-world (Production-world): the object classes and fact types, the result types, and the ontological coexistence rules.

All four aspects models (CM, PM, AM, and SM) constitute the complete ontological knowledge of an organization and the starting point is all available documentation about the enterprise.

The logical sequence of producing the aspect models is anticlockwise, starting with the interaction model (IAM). The first result of the method 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 straight away. It is expressed in an Actor Transaction Diagram (ATD) and a Transaction Result Table (TRT). Next, the Process Structure Diagram (PSD) is produced, and after that the Action Rule Specifications (ARS). The action rules are expressed in a pseudo-algorithmic language. Next, the SM is produced, expressed in an Object Fact Diagram (OFD) and

an Object Property List (OPL). Then, 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).

The general elicitation method to acquire the basis for a correct and complete set of aspect models of an enterprise ontology consists of three analysis and three synthesis steps:

1) The Perfoma-Informa-Forma Analysis (all available pieces of knowledge are divided in three sets, according to the distinction axiom)
2) The Coordination-Actors-Production Analysis (the Performa items are divided into C-acts/results, P-acts/results, and actor roles, according to the operation axiom)
3) The Transaction Pattern Synthesis (for every transaction type, the result type is correctly and precisely formulated; the Transaction Result Table can now be produced)
4) 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)
5) 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)

The purpose of this section is to enlighten what is *DEMO*. Thus, the axioms mentioned aren't discussed.

2.3 Ontological meta-model

The organizational self consists of organizational artifacts (OA). These OA's are arranged in a certain manner as to specific all the spaces (state, process, action and construction) of an organization's world.[3] The OA's have to obey certain rules of arrangement between them. The conceptualization of these rules is called the ontological metal-model of the world. The organization space consists of a set of allowed OA's, thus, the ontological meta-model is the conceptualization of that organization space.

To formulate the ontological meta-model of a world we use WOSL. Therefore we can call it Organization Space Diagram (OSD). In practice, the OSD has its corresponding four DEMO aspect models: SM, CM, PM and AM. The fact that we are dealing with an OSD, they are called, respectively: State OSD, Construction OSD, Process OSD and Action OSD. These diagrams formulate, for each aspect model: the organization artifact kinds out of which instances – OA's – can occur in the organizational self and coexistence rules that govern how to arrange these instances. Figures 6, 7, 8 and 9 presents all four OSD.

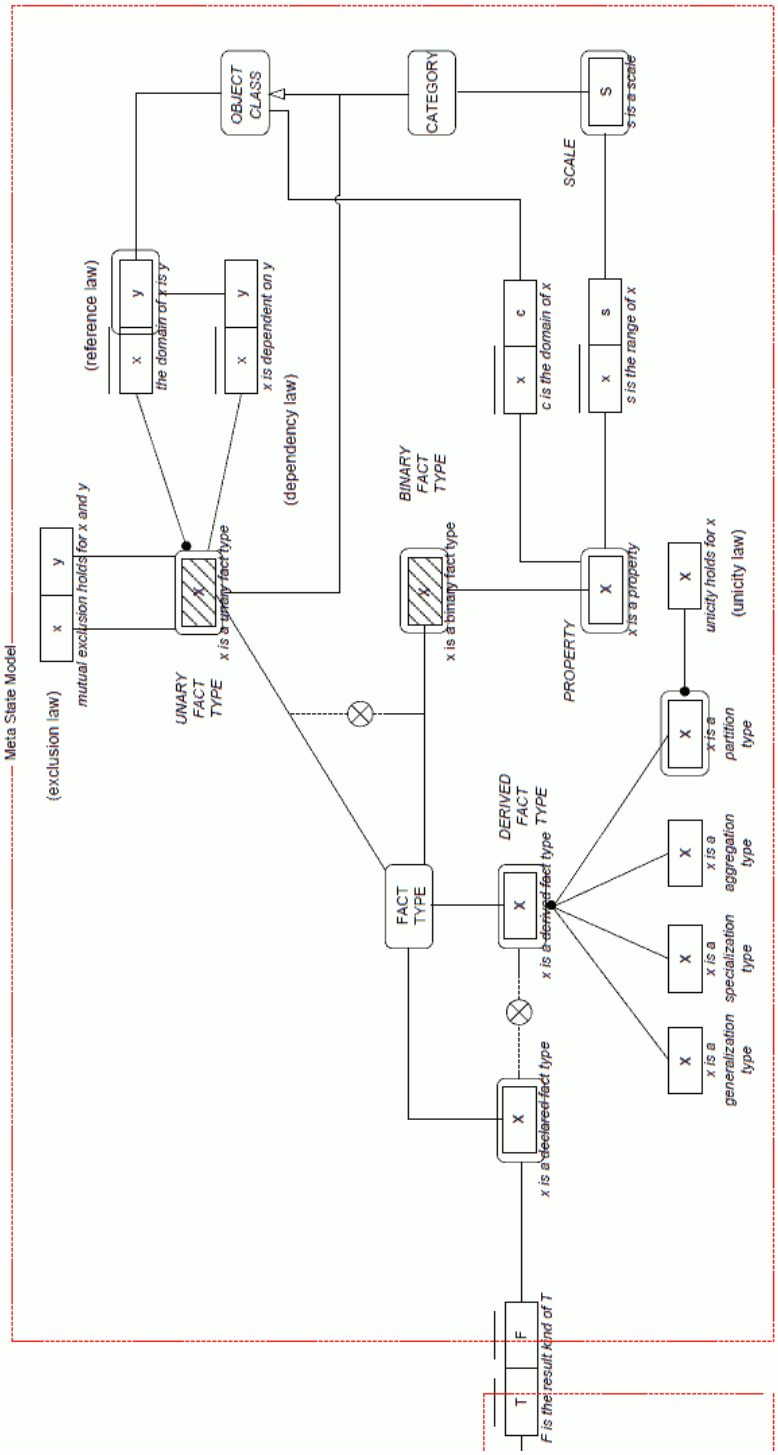

Figure 6: *DEMO* State OSD

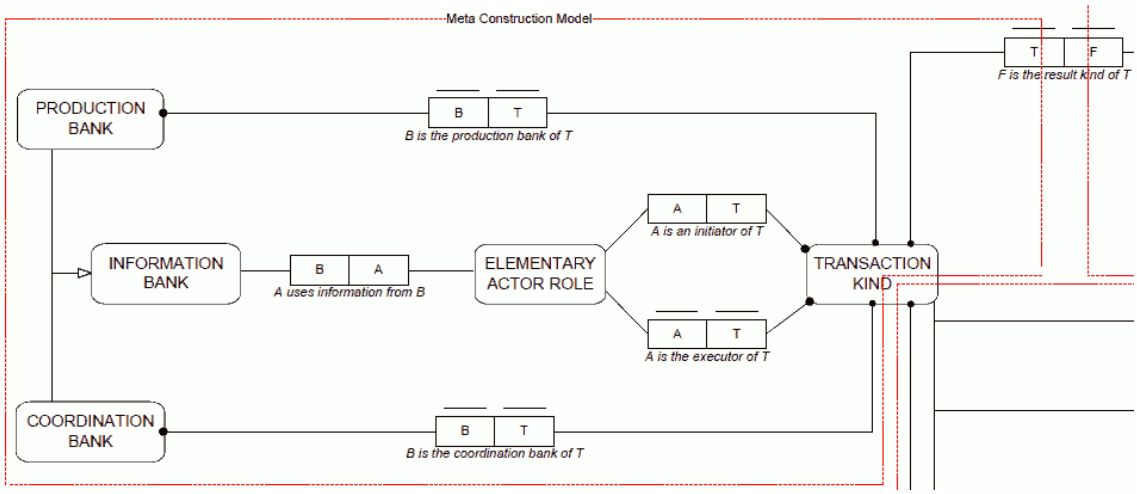

Figure 7: *DEMO* Construction OSD

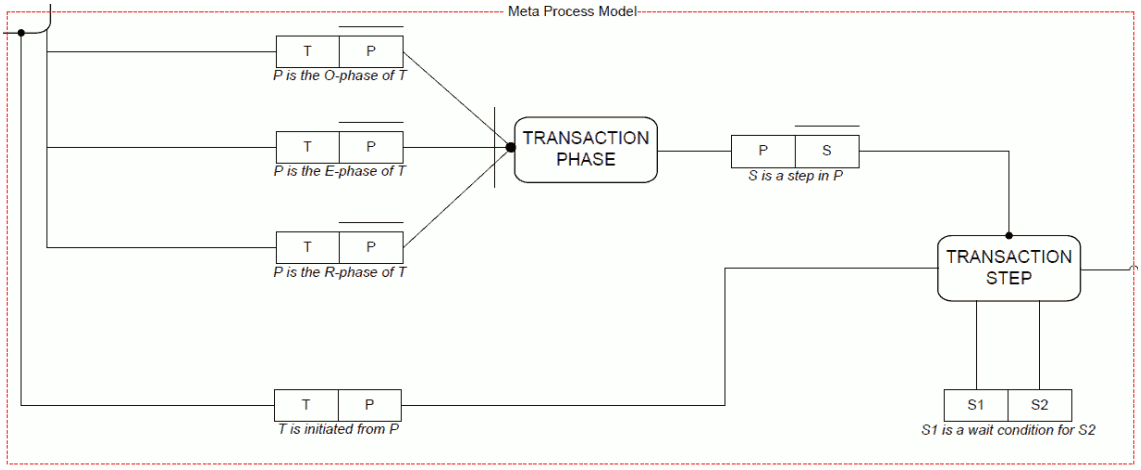

Figure 8: *DEMO* Process OSD

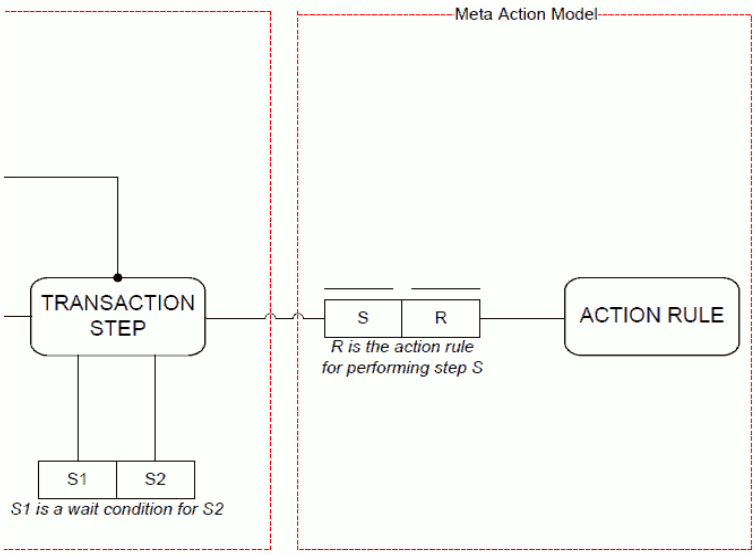

Figure 9: *DEMO* Action OSD

The OSD allows the interpretation of the ontological meta-model. The complete set of organization artifact kinds and laws governing the arrangement of their instances constitutes the organization space. The conceptualization of the organization space consists in the ontological meta-model which, in turn, is formulated in the OSD.

The information presented in the four *DEMO* diagrams will be used to help us define the wiki pages and its semantic properties, because as we said before the ontological meta-model is the conceptualization of an organization space that contains organizational artifacts.

2.4 Semantic Web

To date, the World Wide Web has developed most rapidly as a medium of documents for people rather than of information that can be manipulated automatically. By augmenting Web pages with data targeted at computers and by adding documents solely for computers, we will transform the Web into the Semantic Web.[4]

Computers will find the meaning of semantic data by following hyperlinks to definitions of key terms and rules for reasoning about them logically. The resulting infrastructure will spur the development of automated Web services such as highly functional agents.

Ordinary users will compose Semantic Web pages and add new definitions and rules using off-the-shelf software that will assist with semantic markup.

Definition

The Semantic Web is an evolving extension of the World Wide Web in which the semantics of information and services on the web is defined, making it possible for the web to understand and satisfy the requests of people and machines to use the web content. It derives from World Wide Web Consortium director Sir Tim Berners-Lee's vision of the Web as a universal medium for data, information, and knowledge exchange.

At its core, the semantic web comprises a set of design principles, collaborative working groups, and a variety of enabling technologies. Some elements of the semantic web are expressed as prospective future possibilities that are yet to be implemented or realized.

Other elements of the semantic web are expressed in formal specifications. Some of these include Resource Description Framework (RDF), a variety of data interchange formats (e.g. RDF/XML, N3, Turtle, N-Triples), and notations such as RDF Schema (RDFS) and the Web Ontology Language (OWL), all of which are intended to provide a formal description of concepts, terms, and relationships within a given knowledge domain.

Purpose

Humans are capable of using the Web to carry out tasks such as finding the Finnish word for "monkey", reserving a library book, and searching for a low price on a DVD. However, a computer cannot accomplish the same tasks without human direction because web pages are designed to be read by people, not machines. The semantic web is a vision of information that is understandable by computers, so that they can perform more of the tedious work involved in finding, sharing and combining information on the web.

In 1999, Tim Berners-Lee originally expressed the vision of the semantic web as follows: “I have a dream for the Web [in which computers] become capable of

analyzing all the data on the Web – the content, links, and transactions between people and computers. A ‘Semantic Web’, which should make this possible, has yet to emerge, but when it does, the day-to-day mechanisms of trade, bureaucracy and our daily lives will be handled by machines talking to machines. The ‘intelligent agents’ people have touted for ages will finally materialize.”

Semantic publishing will benefit greatly from the semantic web. In particular, the semantic web is expected to revolutionize scientific publishing, such as real-time publishing and sharing of experimental data on the Internet.

Tim Berners-Lee has described the semantic web as a component of Web 3.0.

2.5 Semantic Wikis

A semantic wiki is a wiki that has an underlying model of the knowledge described in its pages. Regular wikis have structured text and untyped hyperlinks. Semantic wikis allow the ability to capture or identify further information about the pages' (metadata) and their relations.[4]

Key characteristics

Reliance on Formal Notation

The knowledge model found in a semantic wiki is typically available in a formal language, so that machines can process it into an entity-relationship or relational database. The formal notation may be included in the pages themselves by the users, as in *Semantic MediaWiki*. Or, it may be derived from the pages or the page names or the means of linking. For instance, using a specific alternative page name might indicate a specific type of link was intended. This is especially common in wikis devoted to code projects. In either case, providing information through a formal notation allows machines to calculate new facts (e.g. relations between pages) from the facts represented in the knowledge model.

Enables Semantic Web

The technologies developed by the Semantic Web community provide one basis for formal reasoning about the knowledge model that is developed.

Example

Imagine a semantic wiki devoted solely to foods. The page for an apple would contain, in addition to standard text information, some machine-readable semantic data. The most basic kind of data would be that an apple is a kind of fruit – what is known as an inheritance relationship. The wiki would thus be able to automatically generate a list of fruits, simply by listing all pages that are tagged as being of type "fruit." Further semantic tags in the "apple" page could indicate other data about apples, including their possible colors and sizes, nutritional information and serving suggestions, and any other data that was considered notable. These tags could be derived from the text but with some chance of error - accordingly they should be presented alongside that data to be easily corrected.

If the wiki exports all this data in RDF or a similar format, it can then be queried in ways a database might - so that an external user or site could, for instance, submit a query to get a list of all fruits that are red and can be baked in a pie.

Use in knowledge management

Where wikis replace older CMS or knowledge management tools, semantic wikis try to serve similar functions: to allow users to make their internal knowledge more explicit and more formal, so that the information in a wiki can be searched in better ways than just with keywords, offering queries similar to structural databases.

Some systems are aimed at personal knowledge management, some more at knowledge management for communities. The amount of formalization and the way the semantic information is made explicit vary. Existing systems range from primarily content-oriented (like *Semantic MediaWiki*) where semantics are entered by creating annotated hyperlinks, via approaches mixing content and semantics in plain text (like *WikSAR* or living ontology), via content-oriented with a strong formal background (like *IkeWiki*), to

systems where the formal knowledge is the primary interest (like *Platypus Wiki*), where semantics are entered into explicit fields for that purpose.

Also, semantic wiki systems differ in the level of ontology support they offer. While most systems export their data as RDF, some even support various levels of ontology reasoning. To conclude, we can make a comparison: semantic wikis extend and improve regular wikis like semantic web extends World Wide Web.

In our project will be using *Semantic MediaWiki* as a database for all organizational artifacts. Also using semantic properties we can relate and query the information stored. Next section reviews the software we used in our project.

2.6 Review of used software

This section introduces the software used as basis for our project; they served as starting point for what we developed. We start with *MediaWiki* - the wiki-system powering Wikipedia, then the study of *Semantic MediaWiki*, a *MediaWiki* extension to make it semantic. Following is a description of three extensions for *Semantic MediaWiki*: HALO, *SemanticGraph* and *SemanticForms*. Concluding this section is diagram drawing software *Graphviz*.

2.6.1 *MediaWiki*

MediaWiki is free server-based software which is licensed under the GNU General Public License (GPL). It's designed to be run on a large server farm for a website that gets millions of hits per day. *MediaWiki* is an extremely powerful, scalable software and a feature-rich wiki implementation, that uses PHP to process and display data stored in its MySQL database. Pages use *MediaWiki*'s wikitext format, so that users without knowledge of XHTML or CSS can edit them easily.[5]

When a user submits an edit to a page, *MediaWiki* writes it to the database, but without deleting the previous versions of the page, thus allowing easy reverts in case of vandalism or spamming. *MediaWiki* can manage image and multimedia files, too, which are stored in the file system. For large wikis with lots of users, *MediaWiki* supports

caching and can be easily coupled with Squid proxy server software.

Originally developed to serve the needs of the free content Wikipedia encyclopedia, today it has also been deployed by companies for internal knowledge management, and as a content management system. Notably, Novell uses it to operate several of its high traffic websites.

2.6.2 *Semantic MediaWiki*

Semantic MediaWiki (SMW) is a free extension of *MediaWiki*. While traditional wikis contain only texts which computers can neither understand nor evaluate, SMW adds semantic annotations that bring the power of the Semantic Web to the wiki.[6]

Introduction

Wikis have become a great tool for collecting and sharing knowledge in communities. This knowledge is mostly contained within texts and multimedia files, and is thus easily accessible for human readers. But wikis get bigger and bigger, and it can be very time consuming to look for an answer inside a wiki. As a simple example, consider the following question a user might have: «What are the hundred world-largest cities with a female mayor?»

Wikipedia should be able to provide the answer: it contains all large cities, their mayors, and articles about the mayor that tell us about their gender. Yet the question is almost impossible to answer for a human, since one would have to read all articles about all large cities first! Even if the answer is found, it might not remain valid for very long. Computers can deal with large datasets much easier, yet they are not able to support us very much when seeking answers from a wiki: Even sophisticated programs cannot yet read and «understand» human-language texts unless the topic and language of the text is very restricted. The wiki's keyword search does not help either in discovering complex relationships.

Semantic MediaWiki enables wiki communities to make some of their knowledge computer-processable, e.g. to answer the above question. The hard problem for the

computer is to find out what the words in a wiki page (e.g. about cities) mean. Articles contain many names, but which one is the current mayor? Humans can easily grasp the problem by looking into a language edition of Wikipedia that they do not understand (Korean is a good start unless you are fluent there). While single tokens (names, numbers,…) might be readable, it's impossible to understand their relevance in the article. Similarly, computers need some help for making sense of wiki texts.

In *Semantic MediaWiki*, editors therefore add «hints» to the information in wiki pages. For example, someone can mark a name as being the name of the current mayor. This is done by editors who modify a page and put some special text-markup around the mayor's name. After this, computers can access this information (of course they still do not «understand» it, but they can search for it if we ask them to), and support users in many different ways.

Where SMW can help

Semantic MediaWiki introduces some additional markup into the wiki-text which allows users to add "semantic annotations" to the wiki. While this first 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. To illustrate this, we provide some examples from the daily business of Wikipedia:

1. Manually generated lists. Wikipedia is full of manually edited listings such as this one. Those lists are prone to errors, since they have to be updated manually. Furthermore, the number of potentially interesting lists is huge, and it's 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.

2. Searching information. Much of Wikipedia's knowledge is hopelessly buried within millions of pages of text, and can hardly be retrieved at all. For example, at the time of this writing, there is no list of female physicists in Wikipedia. When trying to find all women of this profession that are featured in Wikipedia, one has to resort to textual search. Obviously, this attempt is doomed to fail miserably. Note that among the 20 first results, only 5 are about people at all, and that Marie Curie is not contained in the whole

result set (since "female" does not appear on her page). Again, querying in SMW easily solves this problem (in this case even without further annotation, since existing categories suffice to find the results).

3. Inflationary use of categories. The need for better structuring becomes apparent by the enormous use of categories in Wikipedia. While this is generally helpful, it has also led to a number of categories that would be mere query results in SMW. For some examples consider the categories Rivers in Buckinghamshire, Asteroids named for people, and 1620s deaths, all of which could easily be replaced by simple queries that use just a handful of annotations. Indeed, in this example Category:Rivers, Property:located in, Category:Asteroids, Category:People, Property:named after, and Property:date of death would suffice to create thousands of similar listings on the fly, and to remove hundreds of Wikipedia categories.

4. Inter-language consistency. Most articles in Wikipedia are linked to according pages in different languages, and this can be done for SMW's semantic annotation as well. With this knowledge, you can ask for the population of Beijing that is given in Chinese Wikipedia without reading a single word of this language. This can be exploited to detect possible inconsistencies that can then be resolved by editors. For example, the population of Edinburgh at the time of this writing is different in English, German, and French Wikipedia.

5. External reuse. Some desktop tools today make use of Wikipedia's content, e.g. the media player Amarok displays articles about artists during playback. However, such reuse is limited to fetching some article for immediate reading. The program cannot exploit the information (e.g. to find songs of artists that have worked for the same label), but can only show the text in some other context. SMW leverages a wiki's knowledge to be useable outside the context of its textual article.

2.6.3 *Halo* Extension for SMW

The *Halo* extension is an extension to *Semantic MediaWiki* (SMW) and has been developed as a part of Project *Halo* in order to facilitate the use of Semantic Wikis for a large community of users. Main focus of the development was to create tools that

increase the ease of use of SMW features and advertise the immediate benefits of semantically enriched contents.[7]

Features

Halo enhances SMW by providing intuitive graphical interfaces that facilitate the authoring, retrieval, navigation and organization of semantic data in SMW.

It basically comprises:

- Semantic Toolbar, allowing to quickly inspect, create and alter the semantic annotations of a wiki article;
- Advanced Annotation Mode, for annotating contents semantically in a WYSIWYG-like manner without having to cope with the wiki source text;
- Auto-completion, suggesting entities or data existing in the wiki while typing (e.g. when annotating in the wiki-text or filling in input fields);
- Graphical query interface, empowering users to easily compose queries and preview query results with different output formats;
- Ontology browser, enabling intuitive browsing and changing of the wiki's ontology and look up of instance and property information

The Auto-completion feature will be used in our project to help creating the wiki pages that represent organizational facts.

2.6.4 *SemanticGraph* extension for SMW

SemanticGraph is an extension which depends on *Semantic MediaWiki*, *Semantic Forms* and a working *Graphviz* installation, that generates graphs and trees based on semantic and non semantic wiki structures.[8]

There are 4 parser functions provided by the extension. They are:

- {{#smm:...}} - generated mindmap tree from semantic network, category tree or wiki links

- {{#sgraph:...}} - generated dot network graph
- {{#shypergraph:...}} - generated hypergraph graph
- {{#mm2:...}} display an uploaded mind map with dynamic linking to the wiki.

All of them have similar syntax which can be displayed inline by doing e.g. {{#smm:help}} Tree functions generally require a single root resource to be specified, network graph functions can have a comma seperated list.

It should be possible to use the ouput of a {{#ask..}} query as the definition of a resource list for a graph if you specify the 'format=template' and 'link=none' option in the query.

In our project this extension was adapted to generate *DEMO* diagrams using the parser function {{#sgraph:...}}.

2.6.5 *SemanticForms* extension for SWM

SemanticForms is an extension to *MediaWiki* that allows users to add, edit and query data using forms. It is heavily tied in with the *Semantic MediaWiki* extension, and is meant to be used for structured data that has semantic markup. Having *Semantic MediaWiki* installed is a precondition for the *SemanticForms* extension.[9]

Very simply, *SemanticForms* allows you to have forms for adding, editing and querying data on your wiki, without any programming. Forms can be created and edited not just by administrators, but by users themselves.

The main components of *SemanticForms* functionality are form definition pages, which exist in a new namespace, 'Form:'. These are pages consisting of markup code which gets parsed when a user goes to add or edit data. Since forms are defined strictly through these definition pages, users can themselves create and edit forms, without the need for any actual programming.

The *SemanticForms* extension enforces the use of templates in creating semantic data. It does not support direct semantic markup in data pages; instead, all the semantic markup is meant to be stored indirectly through templates. A form allows a user to populate a pre-defined set of templates for a page (behind the scenes, that data is turned into

semantic properties once the page is saved).

Forms can also be used to edit the data in an existing page, and you can enable an 'edit with form' tab to show up on any page; see The 'edit with form' tab.

SemanticForms also supports autocompletion of fields, so users can easily see what the previously-entered values were for a given field. This greatly helps to avoid issues of naming ambiguity, spelling, etc.

Data in a page that doesn't fit into the form, like a free-form text description of the page's subject, isn't ignored when the page is edited with a form; rather, it is placed into a separate input box called "free text".

SemanticForms also provides hooks to let outside code easily define new input types; this is useful for, among other things, new extensions to define input types that use code that they provide. *SemanticForms* also provides other features: a form to create semantic properties, a form to create templates, a form to create user forms, pages that list all the templates and all the user forms on the site, and others.

2.6.6 *Graphviz*

Graphviz (short for Graph Visualization) is a free package of open source tools for drawing graphs.[10]

2.6.6.1 What is *Graphviz*

Graph visualization is a way of representing structural information as diagrams of abstract graphs and networks. Automatic graph drawing has many important applications in software engineering, database and web design, networking, and in visual interfaces for many other domains, among others.[11]

Graphviz software has several main graph layout programs. These layout programs take a DOT file containing descriptions of graphs in a simple text language, and generate diagrams in several formats such as JPG and PNG images and SVG for web pages. It also has many useful features for concrete diagrams, such as options for colors, fonts,

tabular node layouts, line styles, hyperlinks, and custom shapes. Graphs are usually generated from an external data source, but they can also be created and edited manually, either as raw text files or within a graphical editor.

2.6.6.2 *Graphviz* architecture

Graphviz consists of a graph description language named the «DOT language» and a set of tools that can process DOT files generating a variety of outputs formats (PostScript, PDF, SVG, PNG, and JPG):

- dot – "hierarchical" or layered drawings of directed graphs. The layout algorithm aims edges in the same direction (top to bottom, or left to right) and then attempts to avoid edge crossings and reduce edge length.
- neato – "spring model" layouts. Neato attempts to minimize a global energy function, which is equivalent to statistical multi-dimensional scaling. The solution is achieved using stress majorization, though the older Kamada-Kawai algorithm, using steepest descent, is also available.
- fdp – "spring model" layouts similar to those of neato, but does this by reducing forces rather than working with energy. Fdp implements the Fruchterman-Reingold heuristic including a multigrid solver that handles larger graphs and clustered undirected graphs.
- sfdp – multiscale version of fdp for the layout of large graphs.
- twopi – radial layouts, after Graham Wills 97. The nodes are placed on concentric circles depending their distance from a given root node.
- circo – circular layout, after Six and Tollis 99, Kauffman and Wiese 02. This is suitable for certain diagrams of multiple cyclic structures such as certain telecommunications networks.

2.6.6.3 Drawing graphs with dot

There are two types of graph: directed and undirected. The differences between these two types are that directed graphs are declared using the digraph keyword, and undirected graphs simply use the graph keyword. Also, an arrow (->) is used to show relationships between nodes on directed graphs.

Dot draws directed graphs. It reads attributed graph text files and writes drawings. This language describes three main kinds of objects: graphs, nodes and edges. Because we want to generate directed graphs, the main (outermost) graph is digraph. Figure 10 shows a simple DOT example. The graph name is G and the lines that follow create nodes and edges.

```
1:  digraph G {
2:      main -> parse -> execute;
3:      main -> init;
4:      main -> cleanup;
5:      execute -> make_string;
6:      execute -> printf
7:      init -> make_string;
8:      main -> printf;
9:      execute -> compare;
10: }
```

Figure 10: DOT language example

A node is created when its name first appears in the file. An edge is created when nodes are joined by the edge operator ->. In the above example, line 2 makes edges from «main» to «parse», and from «parse» to «execute». Running dot on this file (call it «graph1.dot»)

dot –Tsvg graph1.dot –o graph1.svg

generates the drawing of Figure 11. The command-line option –Tsvg outputs a svg file that can be displayed using a browser with SVG native support (i.e. Opera or Google Chrome).

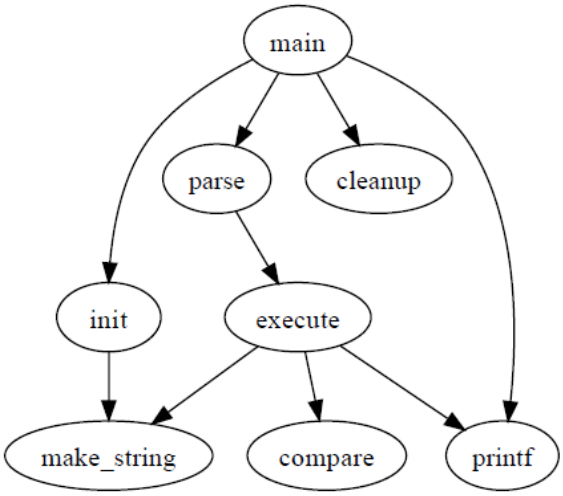

Figure 11: Generated drawing

It is often useful to adjust the representation or placement of nodes and edges in the layout. This is done by setting attributes of nodes, edges, or subgraphs in the input file. [12]

In this project we used dot for all diagram generation and we needed to change several graph, node and edge attributes to obtain the best final diagram.

2.6.6.4 SVG format

Scalable Vector Graphics (SVG) is a family of specifications of an XML-based file format for describing two-dimensional vector graphics, both static and dynamic (i.e. interactive or animated).[13]

The SVG specification is an open standard that has been under development by the World Wide Web Consortium (W3C) since 1999. SVG images and their behaviors are defined in XML text files. This means that they can be searched, indexed, scripted and, if required, compressed. Since they are XML files, SVG images can be created and edited with any text editor, but drawing programs are also available that support SVG file formats.

SVG drawings can be dynamic and interactive and allows three types of graphic objects:

- Vector graphics
- Raster graphics
- Text

Since 2001, the SVG specification has been updated to version 1.1 (current Recommendation) and 1.2 (still a Working Draft).[14]

SVG images, being XML, contain many repeated fragments of text and are thus particularly suited to compression by gzip, though other compression methods may be used effectively. Once an SVG image has been compressed it may be referred to as an "SVGZ" image, with the corresponding filename extension. The resulting file may be as small as 20% of the original size.

SVG images can contain hyperlinks to other documents, using XLink. URLs of SVG images can specify geometrical transforms in the fragment section. An SVG image can define components and use them repeatedly. SVG image can also contain raster graphics (usually PNG and JPEG images) and other SVG images.

The use of SVG on the web is still limited by the lack of support in Internet Explorer which (as of December 2010) is the most widely-used browser. The most widely deployed version of IE (version 8) does not support SVG. However, Microsoft has announced that IE9 will support SVG. Other browsers' implementations are not yet complete. As of 2010, only Opera, Safari and Google Chrome supported embedding via the <img> HTML element. Mozilla Firefox and some other browsers that can display SVG graphics currently need them embedded in <object> or <iframe> elements to display them integrated as parts of an HTML webpage.

There are several advantages to native and full support: plugins are not needed, SVG can be freely mixed with other content in a single document, and rendering and scripting become considerably more reliable.

The main purpose of using SVG files in our project is to easily visualize generated graphs taking advantage of vector images which are composed of a fixed set of shapes, meaning that scaling a vector image preserves the shapes. In simple words we can zoom in and out the SVG image without getting the pixel effect.

2.7 Browsers used

When we started this project the browser used was Internet Explorer 8 (IE8). After the installation of Halo extension IE8 wasn't working with auto-triggered auto-completion. Then we considered using Firefox 3.6.13 which works perfectly with Halo auto-completion. But the problem with Firefox is that a SVG file that contain references to other SVG files (for example, to show the image shape for a transaction) is not properly displayed. Other browsers considered were Opera 11 and Google Chrome 8. These two browsers have the best SVG embedded engine and are great for displaying our SVG files, but they both fail on dealing with auto-triggered auto-completion.

Whatever the browser used we must login as «WikiSysop» and activate auto-triggered auto-completion (see annex A.1, step 30).

2.8 Problems definition

In the beginning of the project some challenges arise and these can be considered our problems.

How to use *Semantic Web* to represent the organizational facts?

Semantic MediaWiki (SMW) is a *MediaWiki* extension. Because it adds semantic relations to wiki entities and helps understanding their context we believe SWM is the starting point to represent organizational facts.

How to define each individual fact?

SMW adds semantic relations to wiki entities but we have to define the best way to formalize those semantic relations (organizational facts).

How to automatically generate DEMO diagrams?

SMW has a strong limitation: information is displayed only with text. Many facets of an organization need specific engineering diagram type. A software tool like *Graphviz* seems appropriate to automatically generate diagrams.

How to link SMW and *Graphviz*?

SMW does not have embedded functions to work with graphs. *MediaWiki* has a huge

community with lots of extensions. We should be able to adapt one of those extensions to serve our purposes. Our first choice was *SemanticGraph* extension.

How to generate better diagrams, in a more efficient way?

By reading and analyzing *SemanticGraph* extension online documentation, we know this is a tool that generates generic diagrams using *Graphviz* software. So is this the extension we are looking for? We should be able to grab the extension frame structure, modify it and add new functions, so that SWM generates better diagrams, in a more efficient way.

2.9 Research strategy

In order to overcome our challenges, a research strategy must be defined based on the problems identified. A solution for representing the organizational facts is to adapt a semantic web software like *Semantic MediaWiki*. A solution for defining each individual organizational fact is creating a wiki page (for each fact) and defining properties that helps us formalize the semantic relations. *Graphviz* software is the solution to automatically generate diagrams through one of its tools, DOT, which can be executed using SMW. Linking SMW and *Graphviz* should be accomplished using *SemanticGraph* extension. This extension retrieves the semantic relations given a set of wiki pages. The solution is to adapt the extension to automatically generate *DEMO* diagrams without any special intervention by the user. We also will be looking to improve the automatic generation of *DEMO* diagrams by creating a new extension that we will name *SemanticDEMO*.

Once introduced the project, its context explained, software used, problems identified and a research strategy defined, next chapter reviews projects related to ours.

3 Related work

In this chapter we study what already has been done in the project area. We did find a project called *Open-Modeling* developed by Jan van Santbrink. This project consists in a web-based application to model and publish architecture models, procedures and related structured information. Not only publishing is done on the web, but modeling and maintaining the information (texts and diagrams) is done using a web browser.[15]

The project URL is: *http://open-modeling.sourceforge.net/*

3.1 *Open-Modeling*

Open-Modeling is a quite complex web-based application that runs over a Tomcat server and Java applets. This open source software supports *DEMO* diagram techniques, and the user can drag symbols, symbol lines that are following the symbols, re-size symbols, color the symbols, and delete symbols.

Figure 12 illustrates the actor roles stored in the *Open-Modeling* database and that are ready to use and generate diagrams.

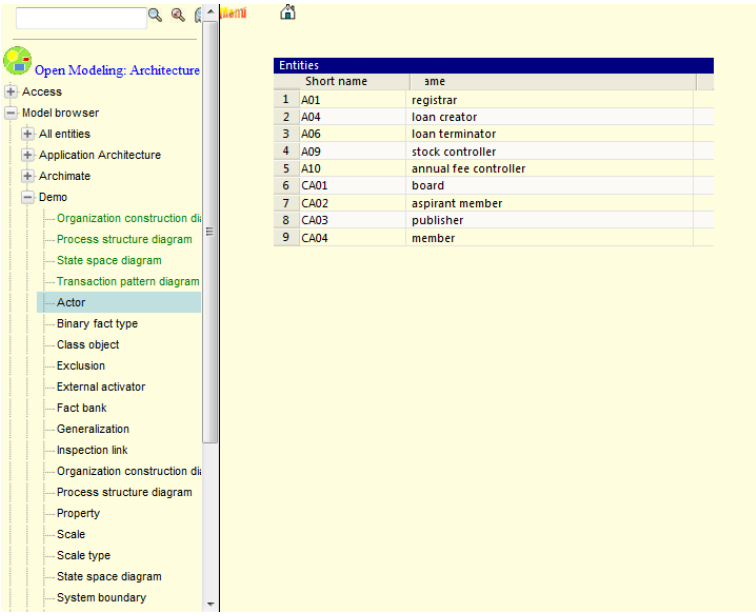

Figure 12: *Open-Modeling* web-based application

Open-Modeling is capable of producing and managing several different types of

models/diagrams. All information is kept in an internal database and models are built on-the-fly when accessing the application. The goal here wasn't to try to build a similar application, but to explore the Java code related to manipulating *DEMO* diagrams and reuse it in our project written in PHP and Javascript.

We can split *Open-Modeling* into two distinct parts: (1) the logical part, that manages and updates the data; (2) and the design part, that grabs the data and produces the different types of diagram. Using an analogy, in our project the logical part will be supported by *Semantic Media Wiki*: by defining semantic properties in the wiki pages and assigning values to these properties, we can store and manage the factual information; the design part will be our main challenge: creating diagrams on-the-fly using the factual information stored in the wiki database.

As we said before, *Open-Modeling* is a quite complex application, and the Java code is very confusing. Thus, probably this will become a limitation to study the code and reuse it properly.

4 Solutions and contributions

With project context acknowledged, problems identified and related applications studied, we now explore the solutions for the problems we elicited. The chapter starts with a review of problems and objectives which then leads us to the solutions.

First is the solution for using *Semantic MediaWiki* to represent organizational facts; this was accomplished by creating wiki pages using a standard nomenclature for those pages and also for the properties embedded.

The second part of the project was finding a way for *Graphviz* to automatically generate *DEMO* diagrams; this was accomplished by modifying *SemanticGraph* extension and using it together with *Graphviz*.

The third part of the project was the creation of a SMW extension named *SemanticDEMO*; this was accomplished by modifying *SemanticGraph* extension and by adding new functions to create and manipulate *DEMO* diagrams.

4.1 Problems and objectives review

At this point it is essential to review the problems defined, and remind our project's objectives.

In summary, our project consists in:

- defining wiki pages and semantic properties using a standard nomenclature to represent organizational fact types;
- using *Halo* extension to use the auto-completion feature through *SemanticForms* extension;
- using *SemanticForms* extension to manage forms that will help us create and edit the semantic properties values;
- in a first phase, modify *SemanticGraph* extension and use it together with *Graphviz* software to generate automatically *DEMO* diagrams;
- mostly important, in a second phase, creating a new extension that we will name

SemanticDEMO to generate automatically *DEMO* diagrams, and manipulate the symbols in a WYSIWYG-like way.

Problems

Use Semantic Web to represent the organizational facts and define each individual organizational fact

Semantic MediaWiki (SMW) adds semantic relations to wiki entities and helps understanding their context. We believe SWM is the starting point to represent organizational facts. SMW adds semantic relations to wiki entities but we have to define the best way to formalize those semantic relations (organizational facts).

Automatically generate *DEMO* diagrams and link SWM to *Graphviz*

SMW has a strong limitation: information is displayed only with text. Many facets of an organization need specific engineering diagram type. SMW does not have embedded functions to work with graphs. *MediaWiki* has a huge community with lots of extensions. We should be able to adapt one of those extensions to serve our purposes. A software tool like *Graphviz* combined with *SemanticGraph* extension seems appropriate to automatically generate diagrams.

Generate and manipulate *DEMO* diagrams in a better and more efficient way

We want to aim higher, and we will be developing a new extension called *SemanticDEMO* (based on *SemanticGraph* extension) to improve the automatic diagram generation and manipulation.

Objectives

Our stating point was *MediaWiki* application and *Semantic MediaWiki* extension. From this base we had our main objectives traced.

The first objective for using *Semantic MediaWiki* to represent organizational facts is to define a standard nomenclature to create pages and properties to prevent inconsistencies in the specification and interpretation of the facts, permitting the most rigorous possible interpretation of concepts present on wiki. A good presentation of the information added

is essential.

The second objective for using *Graphviz* to automatically generate *DEMO* diagrams is to adapt *Semantic MediaWiki* to use *SemanticGraph* extension and *Graphviz* software to help the user's knowledge assimilation by generating diagrams that models an organization. These diagrams should be generated automatically and without any special intervention by the user.

The last objective for creating a new extension is to adapt *SemanticGraph* extension, named *SemanticDEMO* and create a WYSIWYG editor. The goal here is to automatically generate better diagrams because the ones generated using *Graphviz* and *SemanticGraph* extension cannot be fully controlled by the user.

The sections 4.2 and 4.3 contain a description of the applications used to achieve the solution for our problems. They were important because it was through their combination that we overcome the challenges and produce the final result. These sections also contain the solution and how we reached it.

4.2 Using *Semantic MediaWiki* to represent organizational facts

In the next sections we are going to describe how to represent organizational facts. Sections 4.2.1.3 and 4.2.1.4 contains our analysis on how we represent organizational facts using meta-model level concepts and creating a standard nomenclature for wiki pages and properties.

4.2.1 *Semantic MediaWiki*

Semantic MediaWiki has to be adapted for supporting Organizational Engineering. Through the next sub-sections we analyze and show how the applications were used to accomplish the problem solution.

4.2.1.1 *Halo* extension

We decided that auto-completion will be a great help through the task of defining

organizational facts, due the fact that actor roles and transactions have to be adjusted on the fly. Thus, to assist the annotation in *Semantic MediaWiki* we found the *Halo* extension for SMW. The SMW Project *Halo* Extension has been developed as a part of Project *Halo* in order to facilitate the use of Semantic Wikis for a large community of users. Main focus of the developments was to create tools that increase the ease of use of SMW features and advertise the immediate benefits of semantic content. The features of the *Halo* extension can be divided into four main sections:

1) enhancing wiki navigation – features to ease and speed up navigation and access to articles, as well as semantic data, in the wiki
2) improving knowledge authoring – features to allow easy and expressive addition of semantic data to the wiki
3) simplifying knowledge retrieval – features to query knowledge and access information stored in the wiki
4) gardening the knowledge base – features that allow users to detect inconsistencies and continuously improve the quality of the authored knowledge

Halo extension is a very useful extension allowing us to prevent inconsistencies in the specification and interpretation of the facts.

4.2.1.2 Using categories and properties

SMW introduces special markup elements which allow editors to provide «hints» to computer programs on how to interpret some piece of information given in the wiki. Such hints are called semantic annotations and can be viewed as an extension of the existing system of categories in *MediaWiki*.

Categories are an editing feature of *MediaWiki*, and are used as universal "tags" for articles, describing that the article belongs to a certain group of articles (are a means to classify articles according to certain criteria). For example, by adding [[Category:Cities]] to an article, the page is tagged as describing a city. *MediaWiki* can use this information to generate a list of all cities in a wiki, and thus help users to browse the information. You should try to use categories that already exist instead of

creating new ones. Otherwise, the category's article will be empty, and it is strongly recommended to add a description that explains which articles should go into the category. The *MediaWiki* approach is to have many categories on each page, to identify all aspects of that page's subject.

Semantic MediaWiki was created in part to eliminate the need for categories, by allowing for semantic properties to represent this data.[16]

Thus, SMW provides a further means of structuring the wiki. Wiki pages have links and text values in them, but only a human reader knows what the link or text represents. For example, «is the capital of Germany with a population of 3.396.990» means something very different from «plays football for Germany and earns 3.396.990 dollars a year». SMW allows you to annotate any link or text on the page to describe the meaning of the hyperlink or text. This turns links and text into explicit properties of an article. The property «capital of» is different from «on national football team of», just as the property «population» is different from «annual income». This addition enables users to go beyond mere categorization of articles. Properties are used to specify single pieces of information about the topic of some page; the value of a property can either be a standalone value, or the name of a page on the wiki. Every property should be defined on your wiki, with a page in the “Property:” namespace.[17]

Properties are used by a simple mark-up, similar to the syntax of links in *MediaWiki*: [[property name::value]]. This statement defines a «value» for the property of the given «property name». The page where this is used will just show the text for value and not the property assignment.

Consider the Wikipedia article on Berlin. This article contains many links to other articles, such as «Germany», «European Union», and «United States». However, the link to «Germany» has a special meaning: it was put there since Berlin is the capital of Germany. To make this knowledge available to computer programs, one would like to «tag» the link [[Germany]] in the article text, identifying it as a link that describes a «capital property». With SMW, this is done by putting a «property name» and «::» in front of the link inside the brackets, thus: [[Is capital of::Germany]]. In the article, this text still is displayed as a simple hyperlink to «Germany». The additional text «capital of» is the name of the property that classifies the link to Germany. Since categories and

properties merely emphasize a particular part of an article's content, they are often called (semantic) annotations. Information that was provided in an article is now provided in a formal way accessible to software tools.

In our project there is no need to use categories because we don't want to categorize pages. What we need is to be able to represent facts using the ontological meta-model and be able to define some properties regarding each fact (in the context of the meta-model). This is achieved by using properties and assigning values to allow retrieving semantic relations between the organizational facts.

4.2.1.3 Each page represents a fact type at meta-model level

Based on our theoretical research, we decided that each wiki page represents a fact type at meta-model level or a fact at model level. An example of a fact at model level is: «A01-admitter», which is an instance of fact type «ACTOR_ROLE» belonging to the meta-model level (discussed earlier in section 2.3 Ontological meta-model).

For example, the fact that actor roles exist in a certain organization, is represented by a wiki page named «ACTOR_ROLE». This is a fact type which belongs to the meta-model level . We can have instances of this fact type.

For example, the fact that an elementary actor role exists and his name is «admitter» is represented by the wiki page «A01-admitter». This is a fact at model level.

On each page there will be a set of properties that provide semantic to the fact. In the example «A01-admitter», by adding [[is a::ACTOR_ROLE]], the property specifies the meta-model level fact type which is «ACTOR_ROLE». Properties will exist only in the instances of these meta-model level fact types. The only exception is the need to use the property «[[is a::meta_fact_type]]» to state each fact type (at meta-model level) as an instance of itself (a fact at model level), letting us using inline queries to search for all fact types at meta-model level. This is discussed in more detail in section 5.3 .

Thus, each fact type at meta-model level and each fact at model level are properly defined, and by using properties we can create the semantic relations needed for this project.

4.2.1.4 Standard nomenclature for wiki pages and properties

A standard specification is an explicit set of requirements for an item, material, component, system or service.[18]

The need to define a standard nomenclature for wiki pages is crucial to create a homogeneous model and ensure compatibility with other projects that may be developed and integrated with this. A wiki page representing a fact type at meta-model level consists of capital letters and words are separated by underscore. For example, the wiki page for representing a «transaction kind» fact type should be «TRANSACTION_KIND». A wiki page representing an instance of a fact type is a little different. It consists of a capital letter followed by an order number, and then a hyphen, followed by the name of the fact, where words are separated by underscore. For example, the wiki page «A01-admitter» is defined by the capital letter «A» (stands for elementary Actor role) followed by the number «01», and then a hyphen followed by the fact name. On another example, the wiki page «CA02-admission_approver» is defined by the capital letters «CA» (stands for Composite Actor role) followed by the number «02» and then a hyphen followed by the fact name, where the words are separated by underscore.

Properties also have a simple standard nomenclature. Any property consists of lowercase words separated by underscore. Examples of valid properties are: «actor_id», and «initiating_actor_role». The exception is the property «is a»: there is no underscore between the two words. The reason is related to the fact that this is a special property already defined internally in SMW.

4.2.1.5 Support for *DEMO* Methodology

The four aspect models of *DEMO* (Construction Model, Process Model, State Model, and Action Model) are perspectives under the same meta-model. The Construction Model is split in two parts. One of them is the Interaction Model (IAM) which is expressed in an Actor Transaction Diagram (ATD). This part of the organization space can be formulated in the correspondent OSD (which is the *DEMO* Construction OSD) using the ontological meta-model defined in section 2.3 .

To generate an ATD we need to use wiki pages to define the facts represented in the Construction OSD. For now we center our attention on two fact type: «ACTOR ROLE» and «TRANSACTION KIND». They can be represented by two wiki pages following the standard: «ACTOR_ROLE» and «TRANSACTION_KIND». For example, following the standard, instances of these fact types will be: «A01-admitter» and «T02-admission_approval». Because they are instances, properties must be defined. Figure 13 depict the semantic box that shows the properties for the wiki page «A01-admitter».

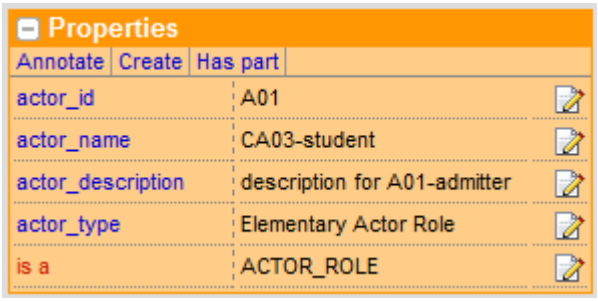

Figure 13: Semantic box for the wiki page «A01-admitter»

In this wiki page we have five properties: «actor_id», «actor_name», «actor_description», «actor_type» and «is a». The property «is a» can be considered a link and creates the semantic relation to the correspondent fact type wiki page («ACTOR_ROLE»). Figure 14 illustrates the textual definition that can be used to create the wiki page «A01-admitter».

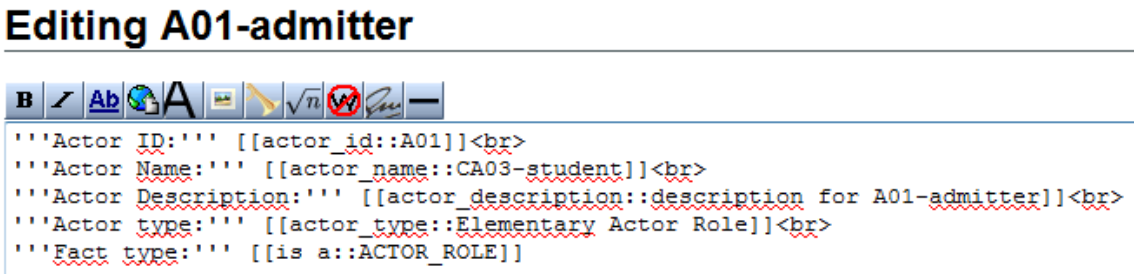

Figure 14: Textual definition for wiki page «A01-admitter»

4.2.1.6 The «DIAGRAM» fact type

To generate a diagram it's useful to know what type of diagram we are going to generate. We've created a wiki page to define the «DIAGRAM» fact type. This fact type

can be used to create instances of «DIAGRAM» fact type.

Any instance of a «DIAGRAM» fact type will have properties. Thus we can set the type through the «diagram_type» property (for example, ATD or OSD) and set the property «fact_type» as «DIAGRAM».

In order to define instances of other fact type («ACTOR_ROLE», «TRANSACTION_KIND», ...), and determine to which diagram they belong, we have defined a property called «Represented in».

4.2.1.7 The «Represented in» property

Now that we have a standard to create the wiki pages, and also can create instances of «DIAGRAM» we need to associate a certain page with one or more diagrams, where a certain graphical object is going to be drawn. To accomplish this goal we've created the «Represented in» property that allow us to define in which diagram(s), for example, a certain instance of «ACTOR_ROLE» fact type can be represented. Also it's useful to store the «x» and «y» coordinates where the graphical object will be drawn in that diagram, where it connects and what connectors are available for other objects.

4.3 Using *Graphviz* to automatically generate *DEMO* diagrams

4.3.1 *SemanticGraph* extension

For generating ATD diagrams we used *SemanticGraph* extension. This extension was created by Rob Challen and is based on *Graphviz* extension for SMW. Basically *SemanticGraph* has seven files, four of which were adapted to serve our purposes. «SemanticGraphSettings.php» holds the essentials configurations for this extension to work properly. First we had to define the complete path for DOT tool, included in the *Graphviz* installation. Then we had to define some options for all dot graphs. These options are related to the graphs, nodes and edges layout (i.e., graph direction left to right, font name and size, minimum distance between nodes and edges, among others).

During the development of this project these options were refined several times until we got the best result.

4.4 Using *SemanticForms* to create wiki pages

In our project, *SemanticForms* extension is used to easily create and manage wiki pages. It allow us to create a template where we define which properties will be held in the wiki pages that will use the template. Each property will be linked to an internal field name. It also allow us to create a form, that will use the defined template. That form can then be used to automatically create wiki pages. The *SemanticForms* extension is also very useful because it allows us to fill the fields with pre-defined data (for example the default data for the field «Represent in» is «ATD1» which refers to the default diagram), set mandatory fields, set hidden fields, and show filling tips for the fields.

4.5 Developing a new extension: *SemanticDEMO*

We used *Graphviz* software and *SemanticGraph* extension to create a diagram. The final diagram was easily readable if few resource pages were defined. On the other hand the diagram was confusing if multiple resource pages were defined. Also we had no control over placing and/or moving the diagram symbols. The main goal for developing a new extension is to overcome these problems and be able to produce better diagrams. The *SemanticDEMO* extension should be able to create a SVG file based on symbol pages that represent instances of fact type defined in our Wiki. This extension should also allow the user to place and/or move diagram symbol, create and/or remove connections and automatically update the Wiki database with the new semantic values.

5 Implementation

This chapter explains in detail the implemented work, the modifications made and where the work was done. It‘s intended to explain project’s details, and especially to help those who will work with the project in the future, clearing up what was done and reporting the problems encountered. Progressing to work done, first part was mostly research and understanding of the project context, and then we had three major topics to work around: how to use *Semantic MediaWiki* to represent organizational facts, how to use *Graphviz* to automatically generate *DEMO* diagrams, and how to create a new extension to generate better *DEMO* diagrams. We begin the chapter with an explanation about *Semantic MediaWiki* inline queries which are related to section 5.2 where we detail how to represent organizational facts using SMW. We also explain how we used *SemanticForms* extension, the «DIAGRAM» fact type and the property «Represented in». Next section explains some concepts related to *Graphviz* and SVG images which are related to section 5.4 where we detail the adaptation of *SemanticGraph* extension to generate *DEMO* diagrams. Then on section 5.5 we summarize the steps needed to automatically generate ATD diagrams. On section 5.6 we detail the implementation of our new extension named *SemanticDEMO*. In the end, some of the problems found are described.

5.1 *Semantic MediaWiki* inline queries

Semantic MediaWiki includes a feature called inline queries that allow users to directly request certain information from the wiki. Inline queries exploit the existing caching mechanisms of *MediaWiki*, thus most requests for a page with such contents can be served without any performance impact whatsoever.[19] The basic way of writing an inline query is to use the parser function #ask. The query string and any printout statements are directly given as parameter, like in the following example:

{{#ask: [[Category:City]] [[located in::Germany]]

| ?population

| ?area#km² = Size in km²

}}

Here we query for all cities located in Germany, and two additional printout statements are used (a simple one and one with some extra settings). This displays the following result on a page:

	Population	Size in km²
Berlin	3,391,409	891.85 km²
Frankfurt		
Munich	1,259,677	310.46 km²
Stuttgart	595,452	208.753 km²

Figure 15: Result query for all cities located in Germany

It is common to put the query as the first parameter behind #ask:. All other parameters are separated by |, just like for other parser functions. The exact formatting of the inline query is not essential, but it is good to use line breaks to make it more readable to other editors: one line per parameter, starting with the | is most accepted in practice.

Note that all the arguments to the #ask: function are ignored by the page parsing, hence the above example does not add a category or a «located in» property annotation to this page. A few more things to note are:

- The pipe '|' symbol is used to separate the conditions from the property to display.
- The conditions for display are a single argument to the #ask function, so there are no '|' symbols between them.
- White space and line breaks can be used within the #ask function, SMW is fairly flexible there.
- The format of the results display changes when you request display of additional properties. SMW picks an appropriate default format for query results, but you also have detailed control of the appearance of query results.

For example, in our project we use inline queries to display all instances of fact type «ACTOR_ROLE».

In the following example the query string is located in the page that represents the fact type:

{{#ask: [[is a::ACTOR_ROLE]] [[actor_type::Elementary Actor Role]]

| ?actor_id

| ?actor_name

| format=table

}}

Here we query for all instances of fact type «ACTOR_ROLE» which are «Elementary Actor Role». This displays the following result on a page:

	Actor id	Actor name
A01-admitter	A01	CA03-student
A03-enroller	A03	enroller
A04-registrar	A04	registrar
A05-course scheduler	A05	course scheduler
A06-course manager	A06	course manager
A07-exam scheduler	A07	exam scheduler
A08-exam manager	A08	exam manager

Figure 16: Result query for fact type «ACTOR_ROLE» which are elementary

The following example differs from the previous because we query for all instances of fact type «ACTOR_ROLE» which are «Composite Actor Role».

{{#ask: [[is a::ACTOR_ROLE]] [[actor_type::Composite Actor Role]]

| ?actor_id

| ?actor_name

| format=table

}}

This displays the following result on a page:

	Actor id	Actor name
CA01-aspirant student	CA01	aspirant student
CA02-admission approver	CA02	admission approver
CA03-student	CA03	student

Figure 17: Result query for fact type «ACTOR_ROLE» which are composite

In our project we also use inline queries to display all fact type at meta-model level. The wiki page «All_Meta_Fact_Type» uses the following inline query:

{{#ask: [[is a::meta_fact_type]]

| ?fact_name

| format=table

}}

This displays the following result on a page:

	Fact name
ACTOR ROLE	ACTOR ROLE
ACTOR ROLE.is an initiator of.TRANSACTION KIND	ACTOR ROLE.is an initiator of.TRANSACTION KIND
ACTOR ROLE.is the executor of.TRANSACTION KIND	ACTOR ROLE.is the executor of.TRANSACTION KIND
TRANSACTION KIND	TRANSACTION KIND

Figure 18: Result query for all fact type at meta-model level

To achieve this result each fact type page has to include the following properties:

'''Fact name:''' [[fact_name::ACTOR ROLE]]

'''Fact type:''' [[is a::meta_fact_type]]

The previous example shows the properties definition for fact type «ACTOR_ROLE».

5.2 Using properties to define organizational facts

As we said before, properties will exist only in the instances of fact types. The only exception is the need to use the property «[[is a::meta_fact_type]]» to state each fact type (at meta-model level) as an instance of itself (a fact at model level). Thus, the wiki

pages «ACTOR_ROLE» and «TRANSACTION_KIND» will hold the «is a::» property and the wiki pages that are instances of these two pages will hold a set of properties. Figure 19 depict the semantic box that shows the properties for the wiki page «CA01-aspirant_student».

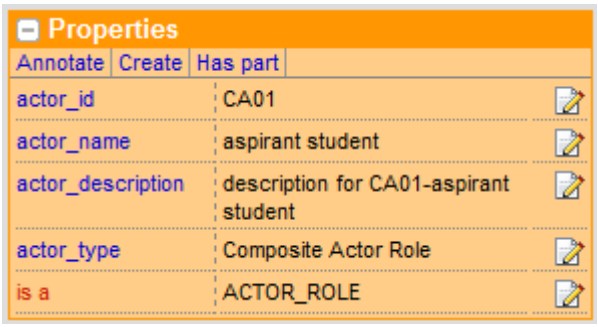

Figure 19: Semantic box for the wiki page «CA01-aspirant_student»

In this wiki page we have five properties: «actor_id», «actor_name», «actor_description», «actor_type» and «is a». The property «is a» can be considered a link and creates the semantic relation to the correspondent fact type wiki page («ACTOR_ROLE»). Figure 20 illustrates the textual definition that can be used to create the wiki page.

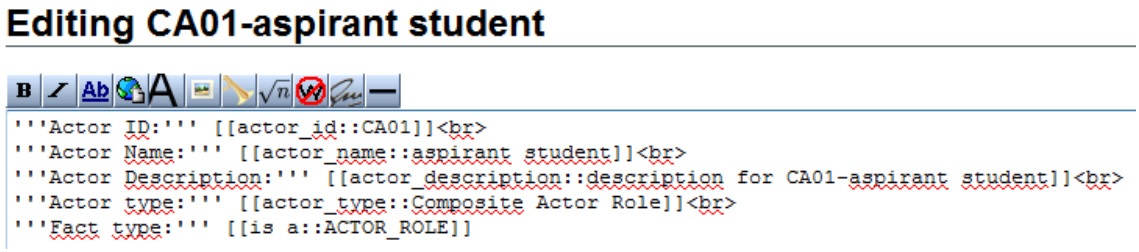

Figure 20: Textual definition for wiki page «CA01-aspirant_student»

Figure 21 depict the semantic box that shows the properties for the wiki page «T02-admission_approval».

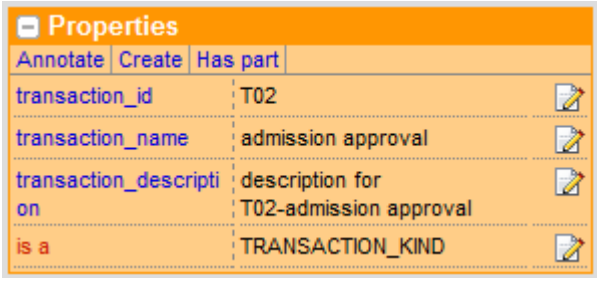

Figure 21: Semantic box for the wiki page «T02-admission_approval»

In this page we have four properties: «transaction_id», «transaction_name», «transaction_description», and «is a». The property «is a» can be considered a link and creates the semantic relation to the correspondent fact type wiki page («TRANSACTION_KIND»). Figure 22 illustrates the textual definition that can be used to create the wiki page «T02-admission_approval».

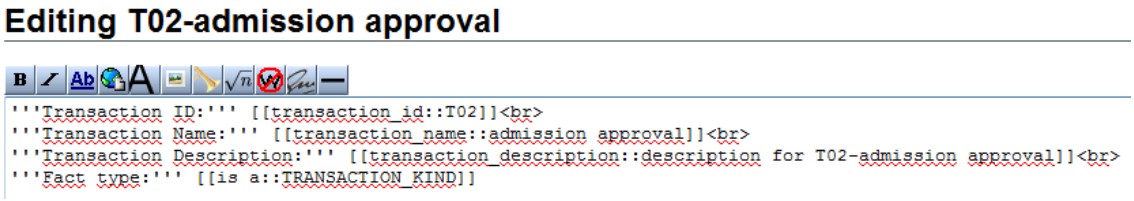

Figure 22: Textual definition for wiki page «T02-admission_approval»

We also want to focus our attention on two binary facts: «A is an initiator of T» and «A is the executor of T». They can be represented by two wiki pages following the standard: «ACTOR_ROLE.is_an_initiator_of.TRANSACTION_KIND» and «ACTOR_ROLE.is_the_executor_of.TRANSACTION_KIND». The reference to the letter «A» means we are dealing with an actor role and the reference to the letter «T» means we are dealing with a transaction kind. The two binary facts can be interpreted replacing those letters. For example, following the standard, instances of these fact types will be: «A01-admitter.is_an_initiator_of.T02-admission_approval» and «A03-enroller.is_the_executor_of.T03-course_enrollment». Figure 23 depict the semantic box that shows the properties for the wiki page «A01-admitter.is_an_initiator_of.T02-admission_approval».

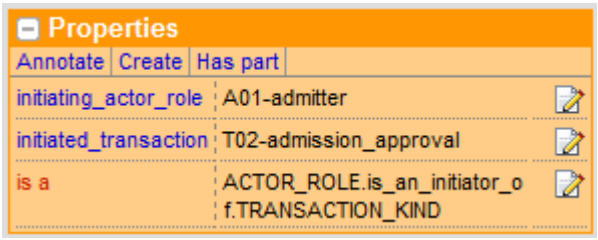

Figure 23: Semantic box for the wiki page «A01-admitter.is_an_initiator_of.T02-admission_approval»

In this page we have three properties: «initiating_actor_role», «initiated_transaction», and «is a». Here the first two properties can be considered as links and creates the semantic relations to the correspondent wiki pages. These properties indicate which actor role initiates the transaction. The property «is a» can be considered a link and creates the semantic relation to the correspondent fact type wiki page («ACTOR_ROLE.is_an_initiator_of.TRANSACTION_KIND»). Figure 24 illustrates the textual definition that can be used to create the wiki page «A01-admitter.is_an_initiator_of.T02-admission_approval».

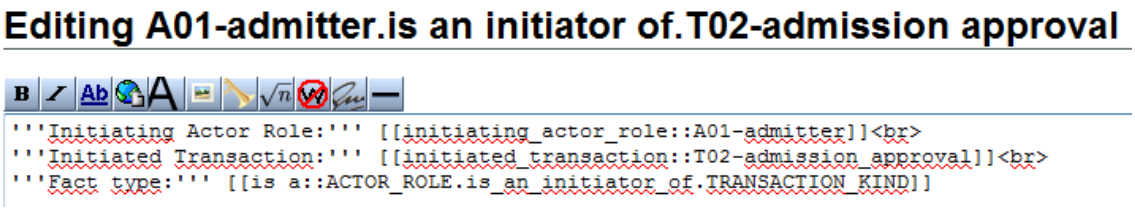

Figure 24: Textual definition for wiki page «A01-admitter.is_an_initiator_of.T02-admission_approval»

Figure 25 depict the semantic box that shows the properties for the wiki page «A03-enroller.is_the_executor_of.T03-course_enrollment».

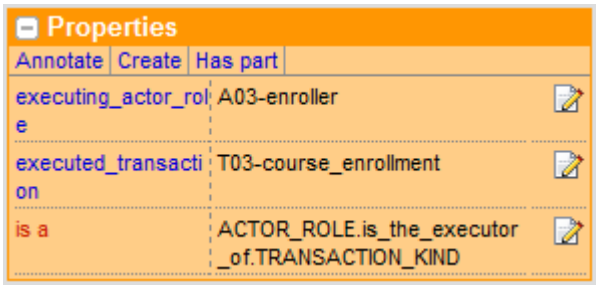

Figure 25: Semantic box for the wiki page «A03-enroller.is_the_executor_of.T03-course_enrollment»

In this page we have three properties: «executing_actor_role», «executed_transaction», and «is a». Here the first two properties can be considered as links and creates the semantic relations to the correspondent wiki pages. These properties indicate which actor role executes the transaction. The property «is a» can be considered a link and creates the semantic relation to the correspondent meta-model fact type wiki page («ACTOR_ROLE.is_the_executor_of.TRANSACTION_KIND»). Figure 26 illustrates the textual definition that can be used to create the wiki page «A03-enroller.is_the_executor_of.T03-course_enrollment».

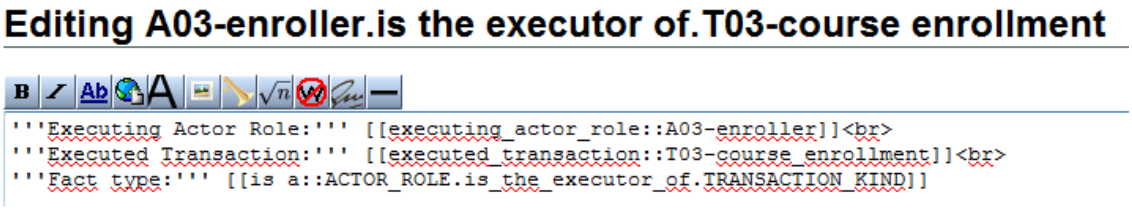

Figure 26: Textual definition for wiki page «A03-enroller.is_the_executor_of.T03-course_enrollment»

The correct way for creating the instances of the fact types is to first create the «ACTOR_ROLE» and «TRANSACTION_KIND» instances and only then create the «ACTOR_ROLE.is_an_initiator_of.TRANSACTION_KIND» and «ACTOR_ROLE.is_the_executor_of.TRANSACTION_KIND».

To help on the task of remembering all the names of the fact types Halo extension is very useful. For example, just start typing «A» and all text starting with that letter will appear on the auto-completion box. Figure 27 shows Halo extension working.

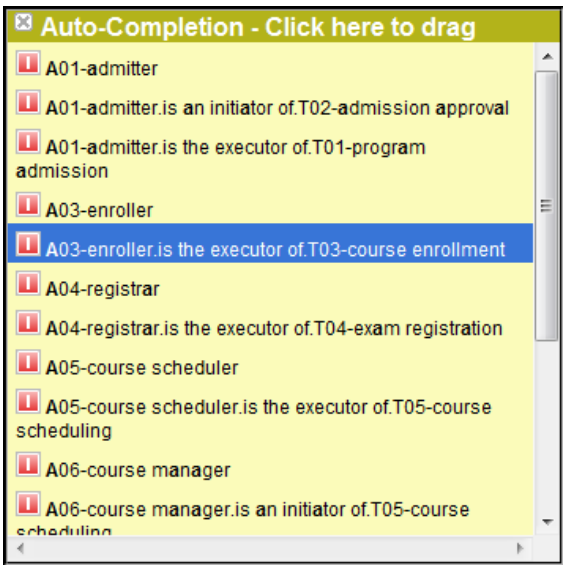

Figure 27: Halo auto-completion box

This way we are able to define properties in a very clear manner to connect pages and create semantic relations between them.

5.2.1 Using *SemanticForms* extension to create wiki pages

In our project, *SemanticForms* extension is used to easily create and manage wiki pages. To use this extension we need to browse «http://localhost/wiki/index.php/Special:SpecialPages», and choose one of the following options:

Figure 28: «SemanticForms» options

First we need to create a template where we define which properties will be held in the wiki pages that will use the template. Each property will be linked to an internal field name. Defining the properties is accomplished using the «Create a template» option.

Figure 29 shows the form used to create a new template.

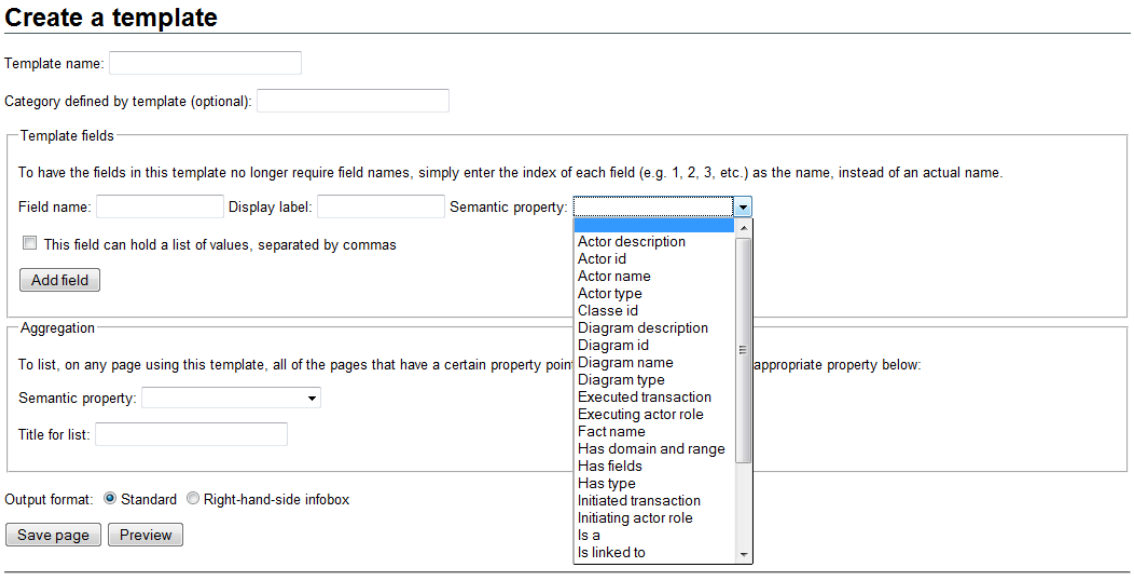

Figure 29: Form used to create a new template

After, we need to create a form using the «Create a form» option (Figure 28).

The next step is to define the form name and which template will be used.

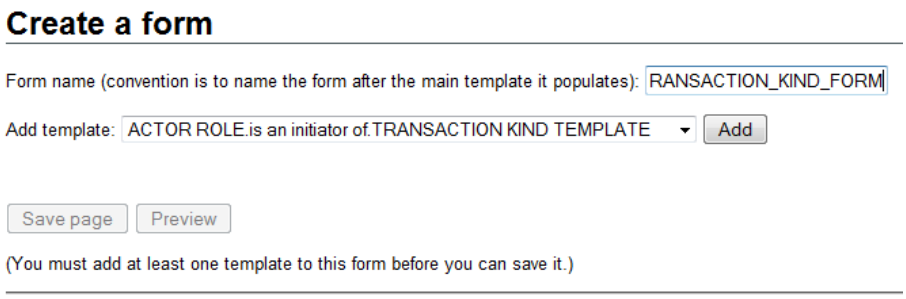

Figure 30: Defining the form name and the template used

In the above figure, we are creating the «ACTOR_ROLE.is_an_initiator_of.TRANSACTION_KIND_FORM» form using the correspondent template (previously-created).

When clicking the «Add» button, the fields and properties defined in the chosen template are loaded and some options can be defined (for example, if a field is mandatory, hidden, and also the property type).

Figure 31 shows the fields, properties and options available in the

«ACTOR_ROLE.is_an_initiator_of.TRANSACTION_KIND_TEMPLATE» template.

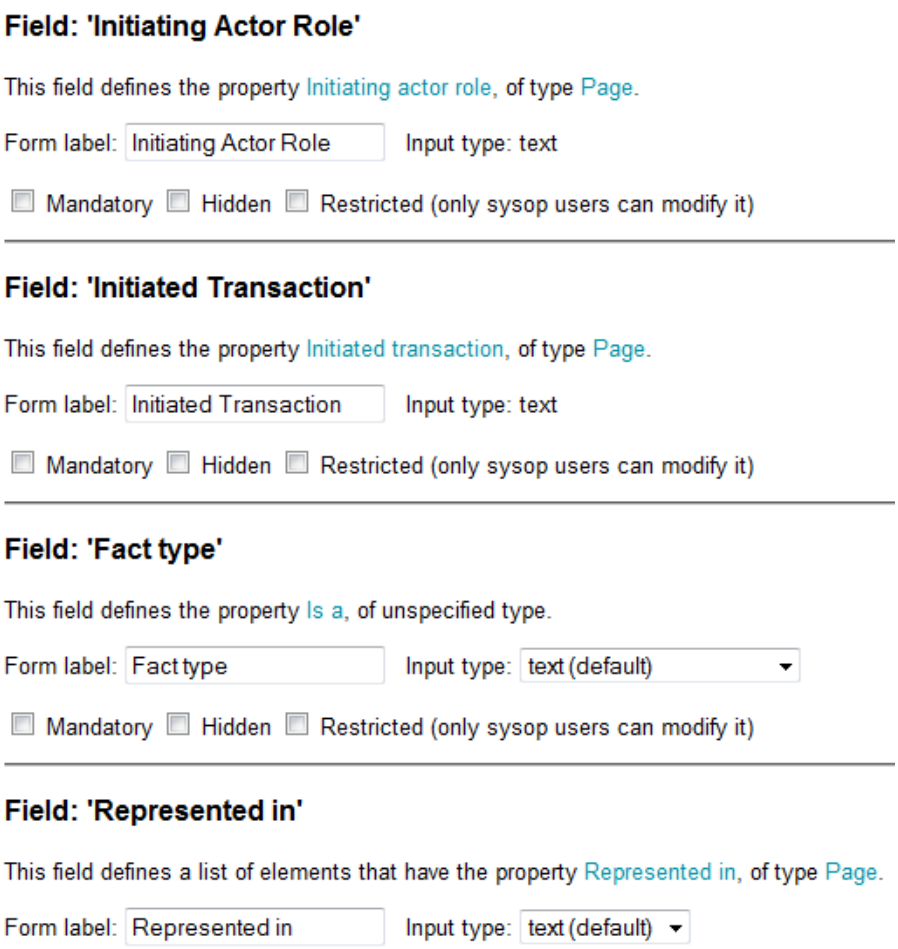

Figure 31: Fields and properties defined in the chosen template

The created form can then be used to automatically create wiki pages that will represent organizational facts at the model level.

We can search for all templates created in our EOMediaWiki by browsing «http://localhost/wiki/index.php/Special:SpecialPages», and choose the «Templates» option:

Figure 32: «Lists of pages» options

Next figure shows 5 templates that we can use to create instances of the correspondent fact types.

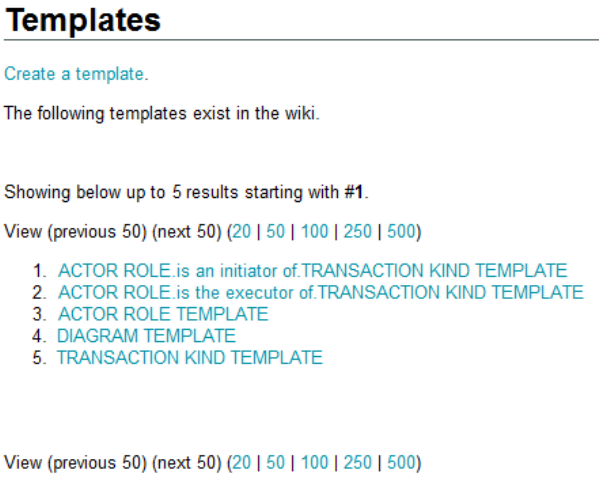

Figure 33: Existing templates in our EOMediaWiki

Clicking on result number 4 (DIAGRAM TEMPLATE) we can access the syntax used to create the correspondent form which will be used to create the wiki page.

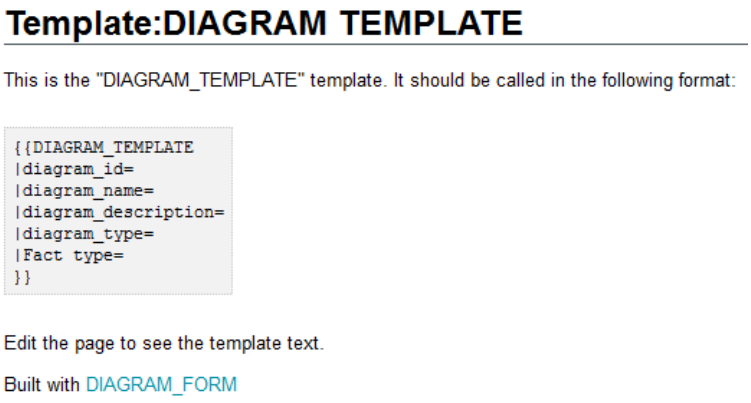

Figure 34: «DIAGRAM TEMPLATE» simple view

Figure 34 shows the simple view. The edit view (template text) is more complex and allows an advanced user to edit and define in detail the template.

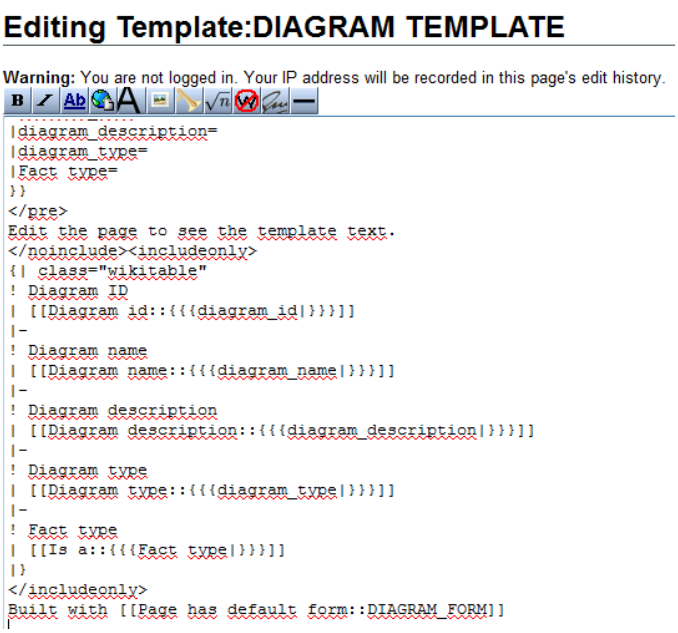

Figure 35: «DIAGRAM TEMPLATE» edit view (template text)

In figure 35 we can observe that each field name is semantically connected to a existing property in EOMediaWiki (for example, the semantic property «Is a» is linked with the field «Fact type»).

In the simple view, we can click «DIAGRAM_FORM» to view the correspondent form.

Figure 36: «DIAGRAM FORM» simple view

Figure 36 shows the simple view. The edit view (form text) is more complex and allows an advanced user to edit and define in detail the form.

Figure 37: «DIAGRAM FORM» edit view (form text)

We can also search for all forms created in our EOMediaWiki by browsing «http://localhost/wiki/index.php/Special:SpecialPages», and choose the «Forms» option.

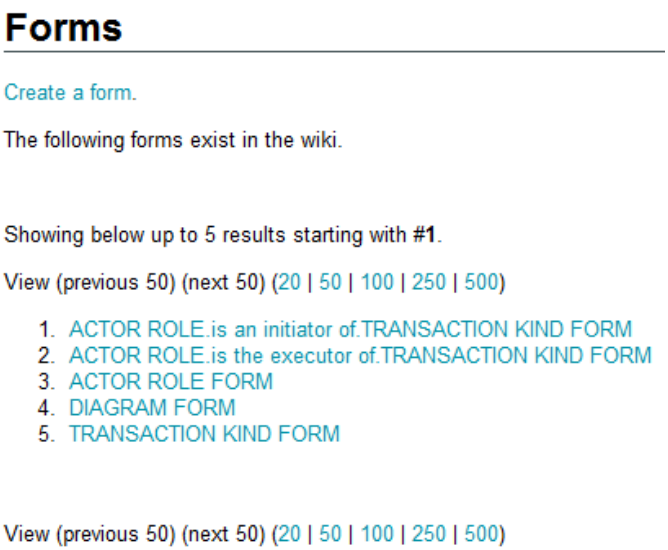

Figure 38: Existing forms in our EOMediaWiki

The process to view and/or edit a form is similar to the template editing process earlier described in detail.

The templates and the fact types have a similar name, for example, «ACTOR_ROLE» and «TRANSACTION_KIND» becomes «ACTOR_ROLE_TEMPLATE» and

«TRANSACTION_KIND_TEMPLATE».

Following the same line of thought, forms and templates also have a similar name, for example, «ACTOR_ROLE_TEMPLATE» becomes «ACTOR_ROLE_FORM».

The *SemanticForms* extension is also very useful because it allows us to fill the fields with pre-defined data (for example the default data for the field «Represent in» is «ATD1» which refers to the default diagram), set mandatory fields, set hidden fields, and show filling tips for the fields.

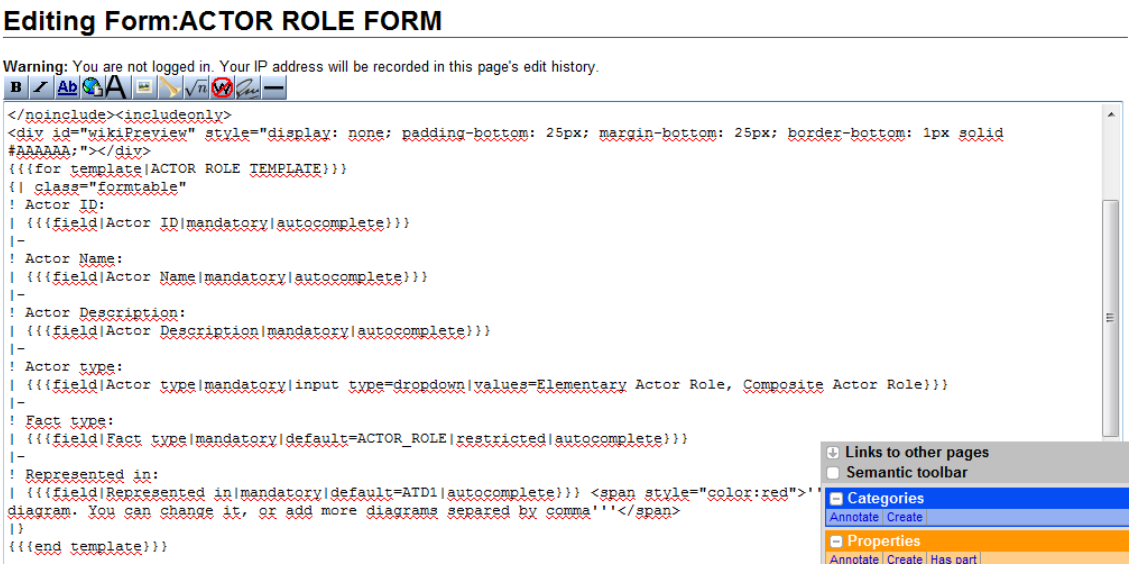

Figure 39: «ACTOR_ROLE_FORM» form

For example, figure 39 shows the «ACTOR_ROLE_FORM» which is based in the «ACTOR_ROLE_TEMPLATE». We can observe that the «Actor type» field/property is mandatory and must be one of «Elementary Actor Role» or «Composite Actor Role». Also the default «Fact type» is «ACTOR_ROLE».

In section 5.2.3 we will address a problem related with the fact that this release version of the extension used in our project does not allow the definition of «n-ary» properties.

5.2.2 Using the «DIAGRAM» fact type

As mentioned in section 5.2.1 , we have created a wiki page to define the «DIAGRAM» fact type. Why the need to create this fact type? The answer is automatically obtained

when looking at the properties defined in the page.

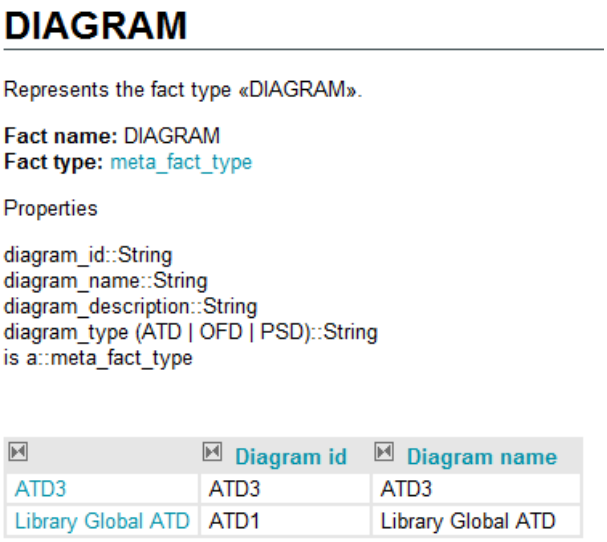

Figure 40: «DIAGRAM» fact type wiki page

Figure 40 shows that this fact type can be used to define diagram wiki pages and imply their type. For example, using a #ask query we obtained a page called «Library Global ATD» with an ID «ATD1». The «Library Global ATD» page definition is shown in the next figure.

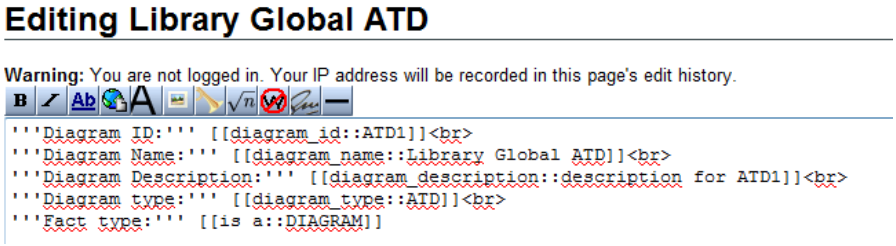

Figure 41: «Library Global ATD» page definition

In figure 41, property «diagram_type» is set as «ATD» and property «fact_type» is set as «DIAGRAM» which implies that this wiki page is a «diagram» fact type.

In order to define instances of other fact type («ACTOR_ROLE», «TRANSACTION_KIND», ...), and determine to which diagram they belong, we have defined a property called «Represented in». This subject is addressed in the next section 5.2.3 .

5.2.3 Using the «Represented in» property

The property «Represented in» is used to define instances of fact type («ACTOR_ROLE», «TRANSACTION_KIND», ...), and determine to which diagram they belong.

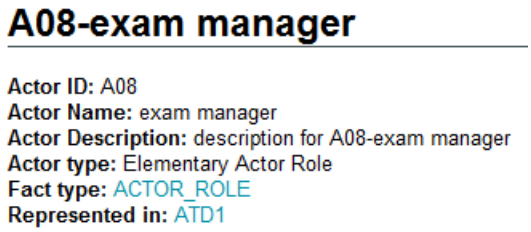

Figure 42: Wiki page «A08-exam_manager» which is an instance of «ACTOR_ROLE» fact type

Figure 42 shows the properties defined in the wiki page «A08-exam_manager» which is an instance of «ACTOR_ROLE» fact type. The property «Represented in» is set to «ATD1» meaning that this instance belongs to the diagram «ATD1».

But this way the wiki page «A08-exam_manager» can only be represented in the diagram «ATD1» and we cannot store any data related to the position of the graphical object in that diagram.

The solution found is to use the «n-ary» properties, now called «records».

The datatype record is used for property values that consist of a short list of values of other datatypes. For each property that uses this datatype, the order and type of the individual fields of the record is fixed, based on a Has fields declaration on the property page. Individual fields in a record value are separated with semicolons (;).

In earlier versions of SMW, record properties have also been called many-valued properties or «n-ary» properties.

A record property is declared by writing [[has type::record]] on the respective property page, and by setting the types of the record's fields with the property Has fields. For example, to define a record with a Date, a Page and a Number field, you can write [[has fields::Date; Page; Number]].[20]

To allow a wiki page to be associated with different diagrams, and define the position of

the object it represents in the diagrams, we redefined the property «Represented in» and used the «record» property as shown in the next figure:

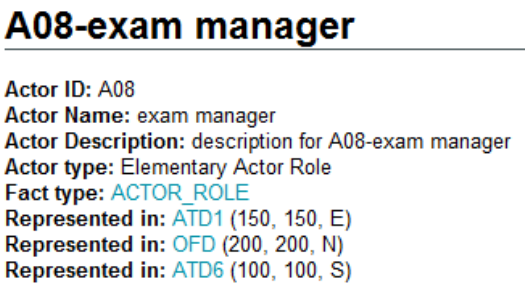

Figure 43: Wiki page «A08-exam_manager» that can be represented in different diagrams

Now the graphical object that the page «A08-exam_manager» represents can be drawn in three different diagrams: ATD1, OFD and ATD6; and on each diagram the object is represented at different coordinates x and y, and connected at the defined point (North, South, East, West).

Figure 44 shows the edit view where we can observe the syntax used. The items are separated by semicolons (;) in front of the «Represented in» property.

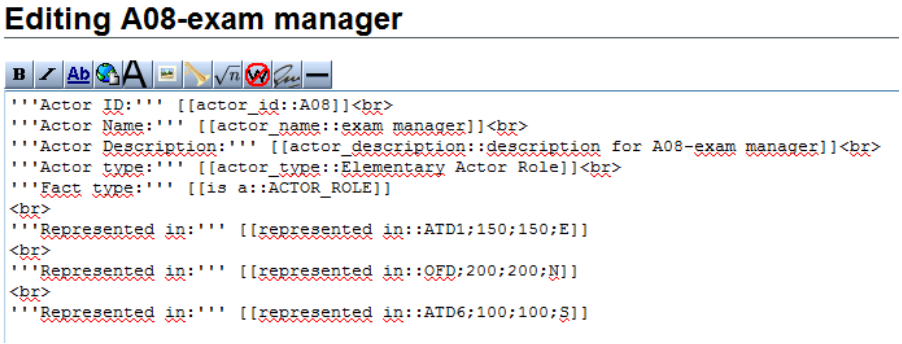

Figure 44: Edit view of wiki page «A08-exam_manager»

Thus, the property «Represented in» is now defined as follows:

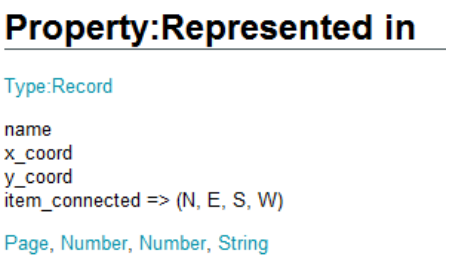

Figure 45: Definition of the «Represented in» property

The property «Represented in» belongs to type «Record», and has four fields: «name» that belong to type «Page», «x_coord» that belongs to type «Number», «y_coord» that belongs to type «Number», and «item_connected» that belongs to type «String».

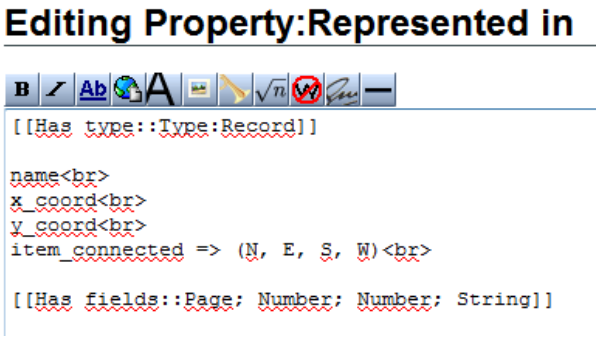

Figure 46: Edit view of the definition of the «Represented in» property

Figure 46 shows the edit view of the property definition. We've managed to set four fields and assign different types using the keyword «[[Has fields:: Page; Number; Number; String]].

The *SemanticForms* release version of the extension used in our project does not allow the definition of «n-ary» properties. Thus, we can still use the extension, create templates and use the forms to easily create and manage wiki pages, but we have to manually edit the page to define the «Represented in» property correctly.

Also related to this subject, we developed a small PHP code that searches EOMediaWiki database to find all wiki pages that must be represented graphically in a certain diagram, based on the «Represented in» property.

First we needed to find the database tables where the data was stored. Thus, the

«eomw_page», «eomw_pagelinks», «eomw_smw_ids», «eomw_smw_rels2» and «eomw_smw_atts2» tables are responsible for storing and relate the semantic properties.

The file «EOMediaWiki_get_diagram_items.php» contains PHP code that allows us to search the database tables and obtain the following result:

Page name	SMW ID	SMW Namespace	Diagram	X coord	Y coord	Conector
A05-course_scheduler	166	323	ATD2	150	150	E

Page name	SMW ID	SMW Namespace	Diagram	X coord	Y coord	Conector
A05-course_scheduler.is_the_executor_of.T05-course_scheduling	182	324	ATD2	150	150	E

Page name	SMW ID	SMW Namespace	Diagram	X coord	Y coord	Conector
A06-course_manager	167	325	ATD2	150	150	E

Page name	SMW ID	SMW Namespace	Diagram	X coord	Y coord	Conector
A06-course_manager.is_an_initiator_of.T05-course_scheduling	184	326	ATD2	150	150	E

Page name	SMW ID	SMW Namespace	Diagram	X coord	Y coord	Conector
A06-course_manager.is_an_initiator_of.T06-course_management	185	330	ATD2	150	150	E

Page name	SMW ID	SMW Namespace	Diagram	X coord	Y coord	Conector
A06-course_manager.is_the_executor_of.T06-course_management	186	331	ATD2	150	150	E

Diagram:

Get items

Try: ATD1, ATD2 or OFD

Figure 47: Search results for the query «ATD2»

Figure 47 shows a HTML table with all wiki pages that are «represented in» ATD2 diagram. For each page we can observe SMW ID and SMW Namespace that are used to obtain the semantic relations. We also observe the properties «diagram», «x_coord», «y_coord» and «connector» that will be used to draw the objects into the diagram. Also the last three properties will be updated when moving the objects using a SVG editor.

This form of association to represented a symbol several times in a diagram arises a issue related to the independence of fact type, instance of fact type and the symbol itself. So we've decided to keep this section to allow the user to see the evolution of the project. To obtain the referred independence we've decided to create a symbol type and use symbol instances, as explained in sections 5.6.1 e 5.6.2 .

5.3 *Graphviz* and SVG images

One of the purposes of using *Graphviz* in our project is to easily generate a graph. But other advantage of using this tool is the possibility to create a SVG file as output. Scalable Vector Graphics (SVG) is a family of specifications of an XML-based file format for describing two-dimensional vector graphics. The vector image is composed of a fixed set of shapes. This means scaling the vector image preserves the shapes. In simple words we can zoom in and out the SVG image without getting the pixel effect. Thus, due to its versatility we decided to use SVG images in our project. In order to create an ATD we needed shapes for actor roles and transactions. We manage to create three SVG images to represent «Composite Actor Role», «Elementary Actor Role», and «Transaction». To create the shapes we used InkscapePortable. Figures 48, 49, and 50 depicts these shapes.

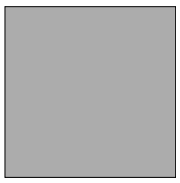

Figure 48: Composite Actor Role shape

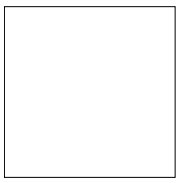

Figure 49: Elementary Actor Role shape

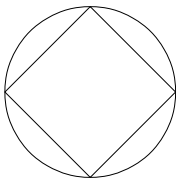

Figure 50: Transaction shape

In our project these images are placed in the «/wiki» server directory under «/images/dot». Also DOT files and SVG files are placed inside that directory. The path to these image files must be defined both on DOT file and SVG file. But there a slightly difference between them. DOT file considers the physical path on the hard drive. On the other hand, SVG file regards the web server URL. He had some problems with file paths. This is discussed in section 5.6 .

SVG 1.1 is a specification of an XML-based file format. Thus, it is an editable file with editable XML tags. By editing the SVG file we managed to add, change and remove some elements (for example, change a physical path to a web server URL).

One useful functionality that we wished for our project is clickable image links. Fortunately, both DOT language and SVG support this feature, and generating a SVG file from a DOT file preserves the URLs. Figure 51 depicts an ATD diagram where

«A04», «T04», «A03», «T03», and «CA03» shapes are clickable links for the correspondent wiki pages.

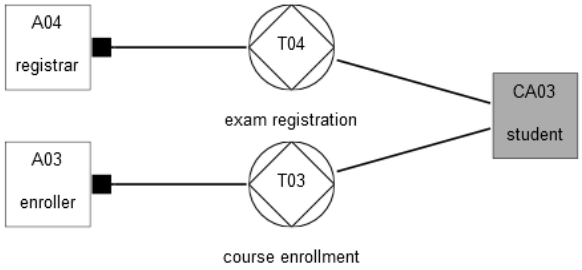

Figure 51: ATD diagram with clickable links

5.4 Adapting the *SemanticGraph* extension

The original *SemanticGraph* extension has seven files. Table 1 shows their filenames, a description about the functions, and indicates the major changes made by us. To simplify the development of new functions and algorithms, all the PHP code related with the generation of graphs as *Freemind* maps and also as *Hypergraph* has been removed. For a more detailed explanation of the changes made please refer to annex A.3.

Filename	Description	Major changes
SemanticGraph2.php	Main file referenced by SMW. Uses «include_once» function to call the other files. Acts like a parser, sets values for some variables and returns an array used for graph render.	Prior to return the array for graph render, we built an algorithm to analyze the array, and add/change/remove DOT elements (i.e. remove duplicate nodes/edges, set shape files for actor roles and transactions). The changed array is used for graph render.
SemanticGraphBuilders.php	Works internally with nodes, child nodes and links from SMW database to start build the graph.	Some functions removed.
SemanticGraphFiles.php	Renders the graph from chosen engine (i.e. dot, neato). Defines file names and file paths. Returns an iframe containing the SVG file (this actually shows the svg graph inside the wiki page).	Renders the graph from dot engine only. Uses DOMDocument to read the temporary svg file. Then adds/changes/removes some XML elements (i.e. places each transaction description below each transaction, changes svg images path), and saves the final svg file. The changed file is viewed using the iframe.
SemanticGraphHelperFunctions.php	Implements functions that allow scrolling and zooming a SVG file.	None (file not used).
SemanticGraphQuery.php	Opens the SWM database and retrieves properties, categories, nodes, and links to create the semantic relations between the wiki pages.	None.
SemanticGraphRenderer.php	Implements a function «enter» that inserts lines into the DOT file, and replaces predefined tags (i.e. «@CHILD_ID@», «@PAR_ID@») by the correspondent arguments passed from «SemanticGraphBuilders.php» file. Removes duplicate lines and breaks long lines using new line character «\n».	Function to break long lines removed. This is done later along with other string functions.
SemanticGraphSettings.php	Holds the settings for the extension (i.e. dot path and command-line, dotoptions). Defines the array template for building the dot file (preamble, nodeExist, nodeNotExist, link, conclusion).	Dot path updated according to our Graphviz installation path. Some changes made in the «dotoptions» array to obtain the best svg graph results.

Table 1: *SemanticGraph* extension files

Shortly, to generate a graph, *SemanticGraph* extension works as follows:

1) in some wiki page we must have the textual definition considering the resource and properties that are going to be used to retrieve the semantic relations. Figure 52 shows an example of how this can be done.

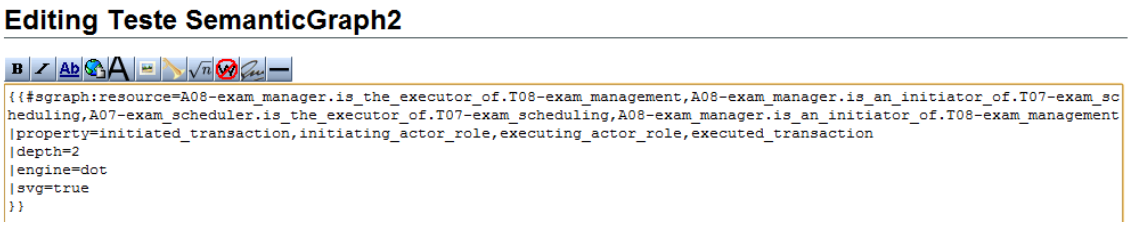

Figure 52: Textual definition for resources and properties

In this example we define four wiki pages as resource and four properties to build the semantic relations.

There are also three arguments: depth (the depth level to search for relations), engine (the render engine to generate the graph) and svg (true meaning that the output will be a SVG file).

2) After clicking the «Save button» a search for semantic relations begins.
3) An array is created based on the template defined and on the nodes, child nodes and links obtained through the semantic search.
4) A DOT file is created based on the array.
5) The DOT file is converted to SVG using *Graphviz* DOT command-line.
6) The SVG file is presented using an iframe in the wiki page.

This generates a graph with the correct semantic relations but visually unattractive because it does not incorporate the ATD graphic shapes.

Thus, before the creation of the DOT file (step 4), the array should be arranged in order to obtain a consistent final result. This is discussed in more detail in section 5.4.1 .

After we obtain a DOT file with all the correct information we can convert it to SVG

format.

Altought we now have a more attractive graph, some elements are still missing. To overcome this problem we used PHP DOMDocument to read the (temporary) SVG file which is build using XML specifications. After some modifications the final SVG is saved and can be presented in the browser iframe. This is discussed in more detail in section 5.4.2 .

5.4.1 Changing DOT file

Before the physical creation of the DOT file, the array should be modified in order to obtain a more consistent final result.

The file «SemanticGraph2.php» was changed to achieve the wanted result. The major changes made are:

- Prior to return the array for graph render, we built an algorithm to analyze the array.
- Then add/change/remove DOT elements (i.e. remove duplicate nodes/edges, set shape files for actor roles and transactions).
- Finally the changed array is used for graph render and creating of the DOT file.

We used some PHP functions to process the array and manipulate strings. Table 2 summarizes the functions and description.

PHP functions	Description
explode	Returns an array of strings, each of which is a substring of string formed by splitting it on boundaries formed by the string delimiter (in DOT language semi-colon is used to end a line, and is the string delimiter).
implode	Returns a string containing a string representation of all the array elements in the same order, with the string delimiter between each element (does the contrary of explode).
preg_match	Searches a string for a match to the regular expression given.
substr	Returns the portion of string specified by the start and length parameters.
strpos	Returns the numeric position of the last occurrence of some expression in a string.
preg_replace	Searches a string for matches to the regular expression given and replaces them with a replacement string.
substr_replace	Replaces a copy of string delimited by the start and length parameters with the string given.
strlen	Returns the length of the given string.
ltrim	Strip whitespace from the beginning of a string.
strcmp	Compares two strings.

Table 2: PHP functions used in the algorithm

The algorithm works as follows:

1) Splits the array
2) Removes from the array all lines containing «.is_an_initiator_of.» and «url» (only the lines containing both)
3) Removes from the array all lines containing «.is_the_executor_of.» and «url» (only the lines containing both)
4) Joins all remaining lines into a unique array
5) Splits the array and searches for all lines containing «label=”T» (means we're dealing with a Transaction)
6) For each transaction gets the transaction name, adds a tooltip with the name, deletes the name from its original location, and add the SVG shape file

7) Searches for all lines containing «label=”A» (means we're dealing with an Elementary Actor role)

8) For each elementary actor role gets the actor name, adds a tooltip with the name, deletes the name from its original location, and add the SVG shape file

9) Searches for all lines containing «label=”CA» (means we're dealing with a Composite Actor role)

10) For each composite actor role gets the actor name, adds a tooltip with the name, deletes the name from its original location, and add the SVG shape file

11) Joins all remaining lines into a unique array

12) Splits the array and searches for all lines containing «.is_the_executor_of.» (the lines that still contain the string are used to create the graph edges)

13) Gets the executor name and strips whitespace from the beginning of the string

14) Compares the executor with the string after “->” and if it's the same string removes the line

15) If they are different adds to the end of the line «[dir=”back”, arrowtail=”box”]»

16) Searches for all lines containing «.is_an_initiator_of.»

17) Gets the initiator name and strips whitespace from the beginning of the string

18) Compares the initiator with the string after “->” and if it's the same string removes the line

19) If they are different adds to the end of the line «[dir=”none”]» (means it's a straight line with no start or end)

20) Joins all remaining lines into a unique array

21) Splits the array and searches for all lines containing «[label=» and adds a quote at the beginning of the string

22) Searches for all lines containing «->» and adds a quote at the beginning and at the end of the string

23) Uses PHP function «array_unique» to remove any duplicate lines that may exist

24) Joins all remaining lines into a unique array

25) Returns the array

At this point the generated diagram should look like Figure 53.

Figure 53: ATD diagram after changing DOT file

The shapes are missing from the diagram due to the path problem explained in section 5.3 . The actor role and transaction labels are also missing. The actor role label should be placed as an extension of «A07» and «A08». The transaction label should be placed below the transaction shape.

The next step is to change the XML elements inside the SVG file to solve these issues.

5.4.2 Changing XML

The graph shapes and most of the labels are missing when the graph is presented. To overcome this problem we used PHP DOMDocument to read the SVG file which is build using XML specifications. At this point this file can be considered a temporary file. After some modifications the final SVG will be saved and presented in the browser iframe. Before the physical creation of the SVG file, some XML tags have to modified by adding or removing some elements and properties.

The file «SemanticGraphFiles.php» was changed to achieve the wanted result and the algorithm works as follows:

1) Uses DOMDocument to load the temporary file to read its contents
2) Gets the «svg» tag and «height» node value
3) Adds 100pt to the height value and updates the node (this expands the image area by 100pt to allow accommodation for transaction labels)

4) Gets all «image» tags and iterates through all of them to get «xlink:href» node string
5) Replaces the current string with a new one and updates the node (this changes the images physical path with the web server path)
6) Gets all «g» tags and iterates through all of them to get «class» node string
7) Gets current «a» tag and for that tag the «title» string
8) Checks if the «class» node string is actually a node
9) Checks if the title string starts with «T», «A», or «CA»
10) If it's a Transaction, gets x and y coordinates of the current «text» tag, and splits the text string into words
11) Counts how many words we've got and for each word creates a new «text» tag containing that word, sets «text-anchor» to middle, the «x» value to the current x coordinate, calculates a new «y» value, and sets the font family and size
12) If it's an Elementary or Composite Actor Role, gets x and y coordinates of the current «text» tag, deletes the text string from it, sets «text-align» to center, and splits the text string into words
13) Counts how many words we've got and for each word creates a new «tspan» tag (inside current «text» tag) containing that word, sets «x» value to the current x coordinate, and calculates a new «y» value
14) Saves the final SVG file

At this point the generated diagram should look like Figure 54.

Figure 54: ATD diagram after changing XML in SVG file

Now the shapes are in place and all text labels are correctly place inside the shapes (for the actor roles) or below the shapes (for transactions).

5.5 Automatic diagram generation: ATD

At this project phase we are capable of generating ATD diagrams with a minimum user intervention. To generate a graph a user has to manually:

1) define wiki page representing facts at model level that must be instances of fact type «ACTOR_ROLE», «TRANSACTION_KIND», «ACTOR_ROLE.is_an_initiator_of.TRANSACTION_KIND», and «ACTOR_ROLE.is_the_executor_of.TRANSACTION_KIND» all belonging to the meta-model level.
2) define a wiki page containing the textual definition considering the resource and properties that are going to be used to retrieve the semantic relations.
3) Save the page mentioned above and wait for the ATD diagram to appear.

The graph generation is an automatic background process described in sections 5.4 , 5.4.1 , and 5.4.2 :

1) semantic relations are acquired based on the resource and properties defined.
2) An array is created based on the template defined and on the nodes, child nodes and links obtained through the semantic search.
3) The array should is arranged in order to obtain a consistent final result.
4) A DOT file is created based on the array.
5) The DOT file is converted to SVG using *Graphviz* DOT command-line.
6) PHP DOMDocument is used to read and modify the (temporary) SVG file.
7) The final SVG file is saved and presented using an iframe in the wiki page.

Although now the process is well defined we had some difficulties that were solved

through research, trial and error, and some practical experience gained in working life.

5.5.1 Testing diagram generation using «Teste_SemanticGraph2» wiki page

The best way to explain in detail how a diagram is generated using *Graphviz* and *SemanticGraph* extension, is to use as an example «Teste_SemanticGraph2» wiki page (http://localhost/wiki/index.php/Teste_SemanticGraph2).

Figure 55 shows «Teste_SemanticGraph2» wiki page in edit mode. We use the «#sgraph» reserved word to load *SemanticGraph* extension and the property «resource» to define which wiki pages are going to be used for generating the diagram (we have to manually define all pages), and «property» to define the linkage point with other wiki pages. We also use the property «engine=dot» to use *Graphviz* DOT engine to process and generate the diagram.

Editing Teste SemanticGraph2

```
{{#sgraph:resource=A08-exam_manager.is_the_executor_of.T08-exam_management,A08-exam_manager.is_an_initiator_of.T07-
exam_scheduling,A07-exam_scheduler.is_the_executor_of.T07-exam_scheduling,A08-exam_manager.is_an_initiator_of.T08-exam_management
|property=initiated_transaction,initiating_actor_role,executing_actor_role,executed_transaction
|depth=2
|engine=dot
|svg=true
|diagram=ATD1
}}
```

Figure 55: «Teste_SemanticGraph2» wiki page in edit mode

After saving the page, SMW processes it using *SemanticGraph* extension. Figure 56 shows the processed «Teste_SemanticGraph2» wiki page.

Teste SemanticGraph2

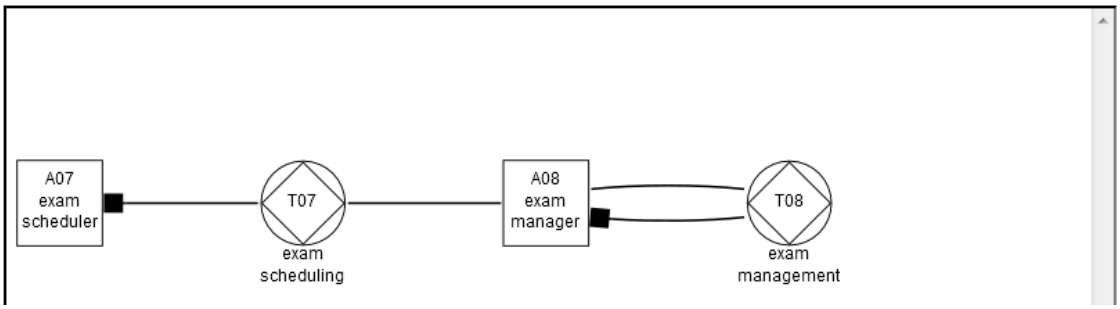

Figure 56: Processed «Teste_SemanticGraph2» wiki page

In edit mode, we've used the property «resource» to which wiki pages are going to be used for generating the diagram:

- A08-exam_manager.is_an_initiator_of.T07-exam_scheduling
- A08-exam_manager.is_an_initiator_of.T08-exam_management
- A07-exam_scheduler.is_the_executor_of.T07-exam_scheduling
- A08-exam_manager.is_the_executor_of.T08-exam_management

The first two pages are instances of «ACTOR_ROLE.is_an_initiator_of.TRANSACTION_KIND» and the other two pages are instances of «ACTOR_ROLE.is_the_executor_of.TRANSACTION_KIND».

All four pages are represented in the diagram using lines: for the first two a straight line, and the last two a straight line ending with a black square. The red circles in figure 57 shows the lines placed in the diagram.

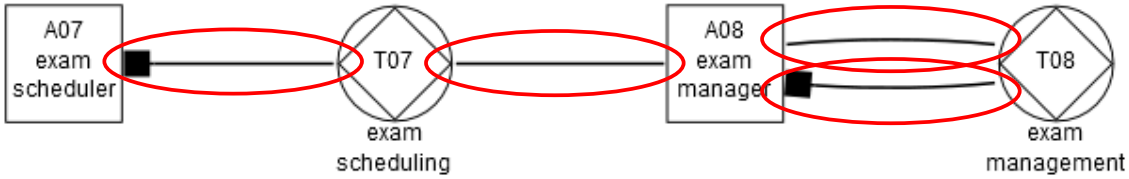

Figure 57: Processed «Teste_SemanticGraph2» diagram

Also in edit mode, we've used the property «property» to define the linkage point with other wiki pages, so *SemanticGraph* extension knows were to obtain that information.

For example, «A08-exam_manager.is_an_initiator_of.T07-exam_scheduling» wiki page has two of those properties defined: «initiating_actor_role» and «initiated_transaction». This tells *SemanticGraph* extension that pages «A08-exam_manager» and «T07-exam_scheduling» are going to be linked. Figure 58 shows the properties definition.

A08-exam manager.is an initiator of.T07-exam scheduling

Initiating Actor Role: A08-exam_manager
Initiated Transaction: T07-exam_scheduling
Fact type: ACTOR_ROLE.is_an_initiator_of.TRANSACTION_KIND
Represented in: ATD1

Figure 58: Properties definition for page «A08-exam_manager.is_an_initiator_of.T07-exam_scheduling»

And another example, «A07-exam_scheduler.is_the_executor_of.T07-exam_scheduling» wiki page has two of those properties defined: «executing_actor_role» and «executed_transaction». This tells *SemanticGraph* extension that pages «A07-exam_scheduler» and «T07-exam_scheduling» are going to be linked. Figure 59 shows the properties definition.

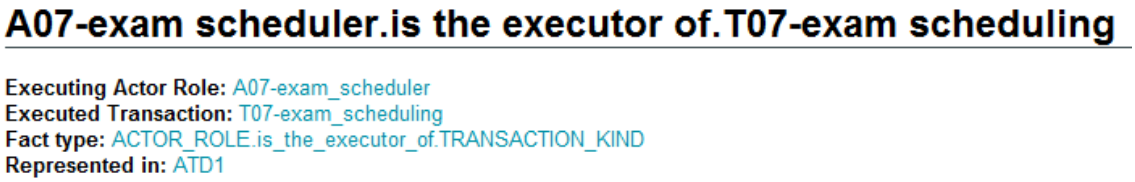

Figure 59: Properties definition for page «A07-exam_scheduler.is_the_executor_of.T07-exam_scheduling»

The definition of these four properties («initiating_actor_role» and «initiated_transaction», «executing_actor_role» and «executed_transaction») are the key to generate the diagram. The red circles in figure 60 shows all four symbols placed in the diagram.

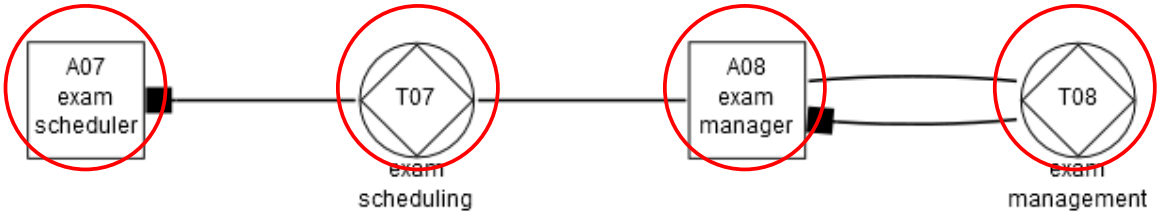

Figure 60: Actor role and Transaction symbols in the final diagram

5.6 Developing *SemanticDEMO* extension

We are capable of generating ATD diagrams with a minimum user intervention but these diagrams were confusing if multiple resource pages were defined. We also wanted to add some new functionality to our project:

- arrange and/or move symbols around a diagram;
- create and/or remove connections between symbols;
- automatically update the Wiki database with the new semantic values.

These can be implemented through the development of a SMW extension, that we are going to name *SemanticDEMO*. The new extension has two folders: «include» and «svg». Table 3 summarizes the «include» folder description. For a more detailed explanation of the changes made please refer to annex A.3.

«include» folder	Description
SemanticDEMO.php	Main file referenced by SMW. Uses «include_once» function to call the other files. Acts like a parser, sets values for some variables and returns a SVG file containing the diagram.
SemanticFunctions.php	Contains several functions, for example: checks the symbol type.

Table 3: «include» folder description

Table 4 summarizes the «svg» folder description.

«svg» folder	Description
Dummy.svg	Contains the SVG basic structure and some Javascript functions.
'diagram'.svg	The diagram created by the extension.

Table 4: «svg» folder description

Before explaining the technical details, we are going to show the practical use of this extension.

First we need to create a wiki page using the following text syntax:

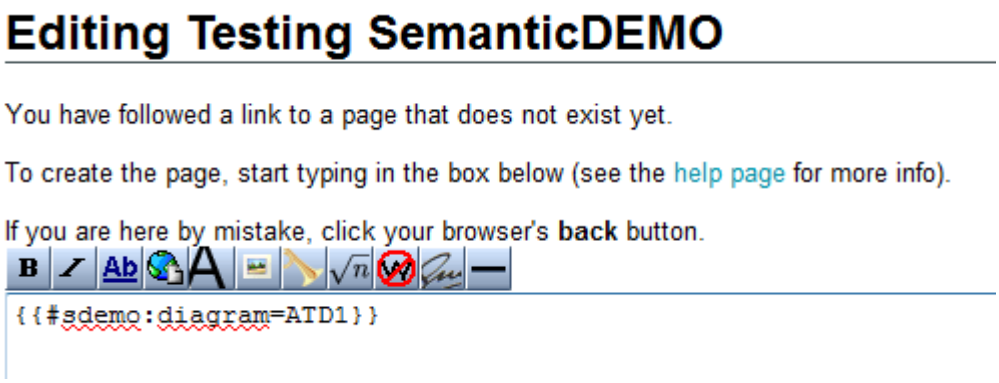

Figure 61: Creating «Testing SemanticDEMO» page

The reserved word «#sdemo» is used to call our extension. The value 'ATD1' is set for the «diagram» property, meaning that all symbol pages with the property «Represented in:», also set with the value 'ATD1', are going to enter the query and placed inside the SVG file.

Figure 62: «Testing SemanticDEMO» page after being processed

Figure 62 shows the saved wiki page, after processing the extension algorithm. To view the SVG file simply follow the link.

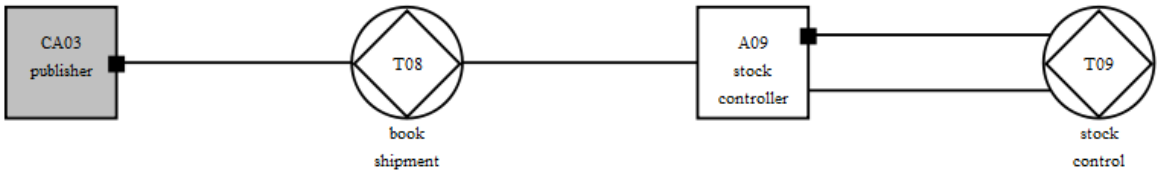

Figure 63: SVG file containing the diagram

At this point the generated diagram should look like Figure 63. The symbols represented in the SVG are obtained from a query to the Wiki database and arranged accordingly to semantic properties.

To accomplishing this phase, we had to make changes in some wiki pages and also create «SYMBOL» fact type pages and symbol pages. The process is described in sections 5.6.1 , 5.6.2 and 5.6.3 .

How the extension works?

We've created a algorithm that basically reads the diagram name defined using the syntax shown in figure 61, and queries wiki database for all pages containing the «Represented in» property whose value is the wanted diagram name. The algorithm then opens a dummy SVG file containing a basic structure, analyzes the symbol type and starts writing in the dummy SVG file. After this process the file is saved using the wanted diagram name. To view the generated SVG file follow the link shown in figure 62.

5.6.1 Creating «SYMBOL» fact type pages

We have created four symbol fact type pages:

- ACTOR_ROLE_SYMBOL
- TRANSACTION_KIND_SYMBOL
- ACTOR_ROLE.is_an_initiator_of.TRANSACTION_KIND_SYMBOL
- ACTOR_ROLE.is_the_executor_of.TRANSACTION_KIND_SYMBOL

By doing this we can now create instances of these fact type pages. For example, «A01-admitter-symbol_01» is a valid instance of «ACTOR_ROLE_SYMBOL». Also «CA01-aspirant_student-symbol_01» is a valid instance of «ACTOR_ROLE_SYMBOL».

5.6.2 Creating symbol instance pages

By creating symbol instance pages we've managed to obtain total independence from the fact type instances earlier defined. For example, «A01-admitter-symbol_01» is independent from «A01-admitter». Also they belong to different fact type: «A01-admitter-symbol_01» is a «ACTOR_ROLE_SYMBOL» instance, and «A01-admitter» is a «ELEMENTARY_ACTOR_ROLE» instance (not a «ACTOR_ROLE» instance, see section 5.6.3).

Each symbol instance page has several properties that allow us to represent it in a

diagram.

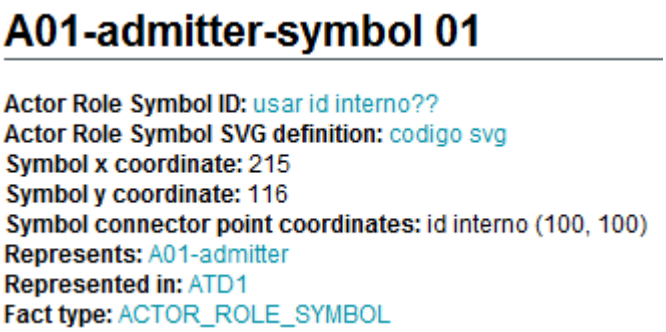

Figure 64: «A01-admitter-symbol_01» properties definition

Figure 64 shows the properties that are used to represent the instance «A01-admitter» in a diagram. This is accomplished using the «Represents» property. Through the property «Represented in» we also reference in which diagram it will be placed, in this case 'ATD1'. We also have the 'x' and 'y' coordinates to draw the symbol in the diagram. This is accomplished using «Symbol x coordinate» and «Symbol y coordinate» properties.

The other properties are still in experimental use. See section 5.6.3 for a complete reference on these properties.

Due the fact that sometimes we have to represent a symbol more than one time in a diagram, we can create 'n' symbol instance pages. For example, to represent the instance «A01-admitter» twice in a diagram, we need to create «A01-admitter-symbol_01» and «A01-admitter-symbol_02» pages. The main difference between them will be the 'x' and 'y' coordinates to draw the symbol in a different position inside the diagram.

5.6.3 Using properties in the symbol pages

During the development of the *SemanticDEMO* extension we had to create «ELEMENTARY_ACTOR_ROLE» and «COMPOSITE_ACTOR_ROLE» fact type. Thus the «ACTOR_ROLE» fact type has become obsolete. But the «ACTOR_ROLE_SYMBOL» is still valid because it still supports the symbol instances.

These new fact type allow us to know the current symbol type. This is important when

drawing the symbol in the SVG.

Table 5 describes the properties present at each symbol instance page.

Property	Description
Actor role symbol ID	The symbol page internal ID (from the wiki database).
Actor role symbol SVG definition	The textual SVG definition to draw the symbol using vector syntax.
Symbol x coordinate	The 'x' coordinate where the symbol will be drawn.
Symbol y coordinate	The 'y' coordinate where the symbol will be drawn
Symbol connector point coordinates	The connector point with the 'x' and 'y' coordinates associated with the correspondent page internal ID.
Represents	The fact type instance that this symbol represents.
Represented in	The diagram in which this symbol is represented.
Fact type	This symbol fact type.

Table 5: Symbol page properties description

The properties explained in table 5 are valid for symbol instances of the following types:

- ACTOR_ROLE_SYMBOL
- TRANSACTION_KIND_SYMBOL

There are slightly changes for symbol instances of «ACTOR_ROLE.is_an_initiator_of.TRANSACTION_KIND_SYMBOL» and «ACTOR_ROLE.is_the_executor_of.TRANSACTION_KIND_SYMBOL».

Figure 65 shows the properties used to represent the symbol in a diagram.

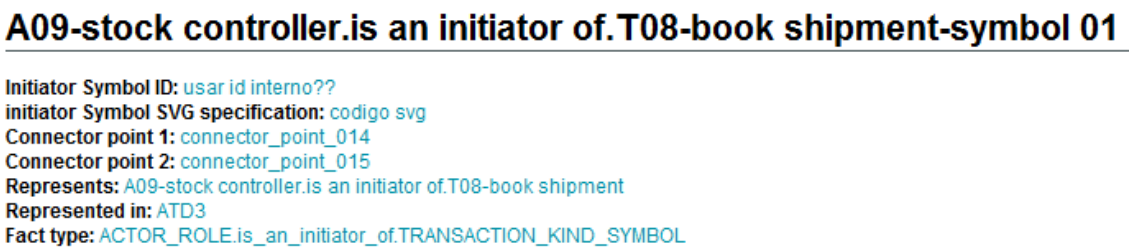

Figure 65: «A09-stock_controller.is_an_initiator_of.T08-book_shipment-symbol_01» properties definition

In this case, since the symbol will be a line, will be needing the 'x' and 'y' coordinates for the start and end of the line. Thus, we use connector points: each connector point contains 'x' and 'y' coordinates.

The connector point definition can be made as shown in figure 66.

Figure 66: «Connector_point_14» properties definition

5.6.4 Testing diagram generation using «ATD3» wiki page

During our project development new wiki pages and new properties were created. We also changed some properties name. The best way to show those changes, and to explain in detail the final result of our project efforts, is to use as an example «ATD3» wiki page (http://localhost/wiki/index.php/ATD3).

Figure 67 shows «ATD3» wiki page in edit mode. We use the «#sdemo» reserved word to load *SemanticDEMO* extension and the property «diagram» to define which diagram we intend to generate. This means that *SemanticDEMO* will grab all symbol pages instances that are associated to this diagram using the «represent in» property («represented in::ATD3»). We also use the «#ask» reserved word to query wiki database and get all symbol pages that are going to be needed for the generation of this diagram.

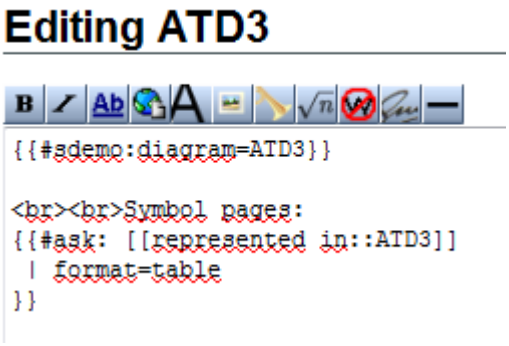

Figure 67: «ATD3» wiki page in edit mode

After saving the page, SMW processes it using *SemanticDEMO* extension. Figure 68 shows the processed «ATD3» wiki page.

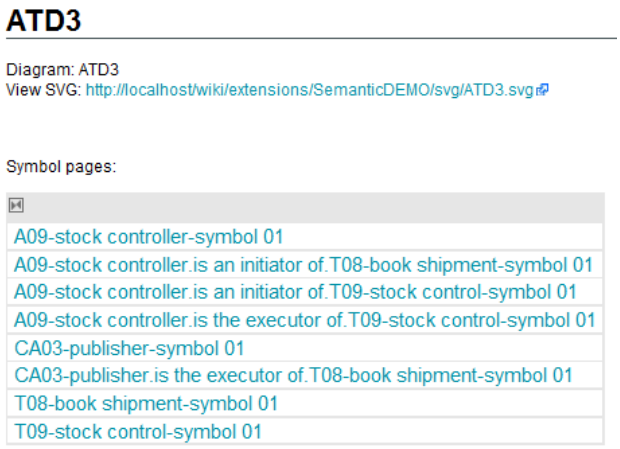

Figure 68: Processed «ATD3» wiki page

SVG diagram file can be loaded by clicking on the link next to «View SVG». Figure 69 shows the generated diagram.

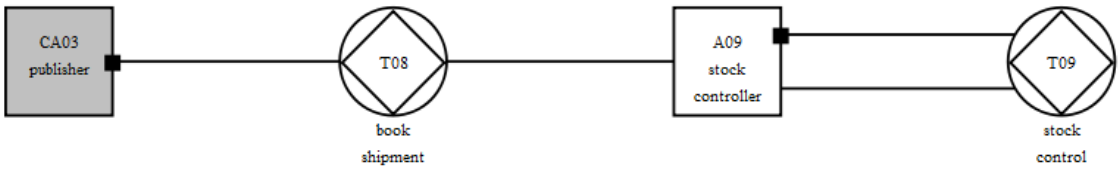

Figure 69: «ATD3» DEMO generated diagram

Back to the processed «ATD3» wiki page, at the page top there is some debug

information used during our tests to ensure that we were retrieving all properties values needed. Figure 70 shows the resulting debug information.

ELEMENTARY_ACTOR_ROLE

Symbol page title	Symbol page ID	X coord - Property SMW ID	X coord	Y coord - Property SMW ID	Y coord	Represents	which is a
A09-stock_controller-symbol_01	405	345	600	346	200	A09-stock_controller	ELEMENTARY_ACTOR_ROLE

COMPOSITE_ACTOR_ROLE

Symbol page title	Symbol page ID	X coord - Property SMW ID	X coord	Y coord - Property SMW ID	Y coord	Represents	which is a
CA03-publisher-symbol_01	404	345	100	346	200	CA03-publisher	COMPOSITE_ACTOR_ROLE

TRANSACTION_KIND

Symbol page title	Symbol page ID	X coord - Property SMW ID	X coord	Y coord - Property SMW ID	Y coord	Represents	which is a
T08-book_shipment-symbol_01	406	345	350	346	215	T08-book_shipment	TRANSACTION_KIND

TRANSACTION_KIND

Symbol page title	Symbol page ID	X coord - Property SMW ID	X coord	Y coord - Property SMW ID	Y coord	Represents	which is a
T09-stock_control-symbol_01	408	345	850	346	215	T09-stock_control	TRANSACTION_KIND

Figure 70: Debug information resulting from querying wiki database

At the page bottom we use a wiki query to obtain all symbol pages used to generate the diagram. To generate this diagram *SemanticDEMO* extension processed eight symbol pages. Some of these symbol pages are instances of the same fact type. For example, «A09-stock controller.is an initiator of.T08-book shipment-symbol 01» and «A09-stock controller.is an initiator of.T09-stock control-symbol 01» represent two distinct symbols, but they both are instances of «ACTOR_ROLE.is_an_initiator_of.TRANSACTION_KIND_SYMBOL» wiki page.

Let's analyze one symbol page per fact type. First we have «A09-stock controller-symbol 01» symbol page (http://localhost/wiki/index.php/A09-stock_controller-symbol_01). Figure 71 shows the edit mode for this page where we can see the properties and values defined.

Editing A09-stock controller-symbol 01

```
'''Actor Role Symbol ID:''' [[actor_role_symbol_id::usar id interno??]]<br>
'''Actor Role Symbol SVG specification:''' [[actor_role_symbol_svg_spec::codigo svg]]<br>
'''Symbol x coordinate:''' [[x_coordinate::600]]<br>
'''Symbol y coordinate:''' [[y_coordinate::200]]<br>
'''Symbol connector point coordinates:''' [[connector_point_coordinates::id interno;100;100]]<br>
'''Represents:''' [[represents::A09-stock controller]]<br>
'''Represented in:''' [[represented_in::ATD3]]<br>
'''Fact type:''' [[is a::ACTOR_ROLE_SYMBOL]]
```

Figure 71: «A09-stock controller-symbol 01» symbol page

At this point, *SemanticDEMO* extension uses only the properties listed in table 6:

Property	Value	Description
x_coordinate	600	Draws the symbol in x coordinate.
y_coordinate	200	Draws the symbol in y coordinate.
represents	A09-stock controller	Represents elementary actor role named «stock controller»; there is a wiki page to represent it.
represented in	ATD3	Draws the symbol in ATD3 diagram; there is a wiki page to represent it.
is a	ACTOR_ROLE_SYMBOL	This symbol is an instance of ACTOR_ROLE_SYMBOL

Table 6: «A09-stock controller-symbol 01» properties

«A09-stock controller» is a wiki page that represents an elementary actor role named «stock controller». Figure 72 shows the properties and values assigned to this page.

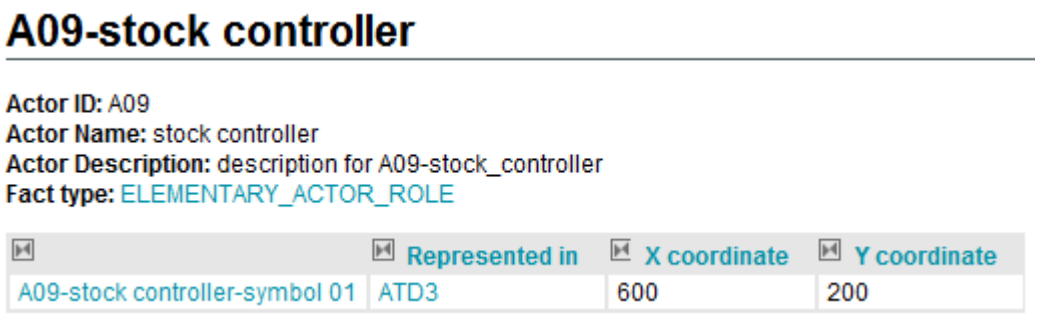

Figure 72: «A09-stock controller» wiki page

We've defined «A09-stock controller» as an «ELEMENTARY_ACTOR_ROLE» instance, and we've built a query to obtain all symbol pages that can represent it, because this actor role can be used in several different diagrams. In this case there is only one symbol page, that will draw the symbol at coordinates (600,200) in diagram ATD3. The red circle in figure 73 shows the symbol placed in the diagram.

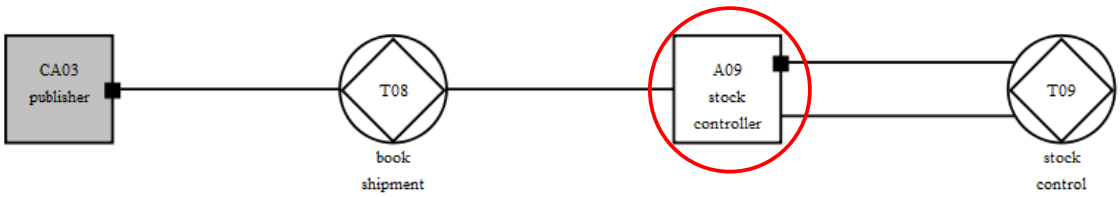

Figure 73: «A09-stock controller» symbol

Back to the processed «ATD3» wiki page, we also have «A09-stock controller.is an initiator of.T08-book shipment-symbol 01» symbol page (http://localhost/wiki/index.php/A09-stock_controller.is_an_initiator_of.T08-book_shipment-symbol_01). Figure 74 shows the edit mode for this page where we can see the properties and values defined.

Editing A09-stock controller.is an initiator of.T08-book shipment-symbol 01

```
'''Initiator Symbol ID:''' [[initiator_symbol_id::usar id interno??]]<br>
'''initiator Symbol SVG specification:''' [[initiator_symbol_svg_spec::codigo svg]]<br>
'''Connector point 1:''' [[connector_point_1::connector_point_014]]<br>
'''Connector point 2:''' [[connector_point_2::connector_point_015]]<br>
'''Represents:''' [[represents::A09-stock controller.is an initiator of.T08-book shipment]]<br>
'''Represented in:''' [[represented_in::ATD3]]<br>
'''Fact type:''' [[is a::ACTOR_ROLE.is_an_initiator_of.TRANSACTION_KIND_SYMBOL]]
```

Figure 74: «A09-stock controller.is an initiator of.T08-book shipment-symbol 01» symbol page

SemanticDEMO extension uses only the properties listed in table 7:

Property	Value	Description
connector_point_1	connector_point_014	Links to the wiki page containing x and y coordinates defined for point 1.
connector_point_2	connector_point_015	Links to the wiki page containing x and y coordinates defined for point 2.
represents	A09-stock controller.is an initiator of.T08-book shipment	Represents elementary actor role named «stock controller» is an initiator of transaction named «book shipment»; there is a wiki page to represent it.
represented in	ATD3	Draws the symbol in ATD3 diagram; there is a wiki page to represent it.
is a	ACTOR_ROLE.is_an_initiator_of.TRANSACTION_KIND_SYMBOL	This symbol is an instance of ACTOR_ROLE.is_an_initiator_of.TRANSACTION_KIND_SYMBOL

Table 7: «A09-stock controller.is an initiator of.T08-book shipment-symbol 01» properties

This symbol page draws a line. Thus, «connector_point_014» and «connector_point_015» are wiki pages that represent the x and y coordinates of the start and end of that line. Figures 75 and 76 shows the properties and values assigned to those pages.

Figure 75: «connector_point_014» wiki page

Figure 76: «connector_point_015» wiki page

In this case, the connector starts at coordinates (430, 240) and ends at coordinates (600,240). The red circle in figure 77 shows the symbol placed in the diagram.

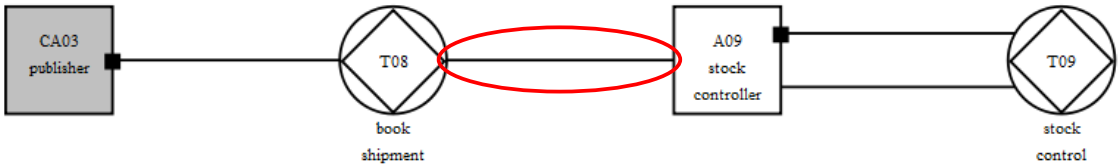

Figure 77: «A09-stock controller.is an initiator of.T08-book shipment» symbol

«A09-stock controller.is an initiator of.T08-book shipment» is a wiki page that represents an instance of «ACTOR_ROLE.is_an_initiator_of.TRANSACTION_KIND». Figure 78 shows the properties and values assigned to this page.

A09-stock controller.is an initiator of.T08-book shipment

Initiating Actor Role: A09-stock_controller
Initiated Transaction: T08-book_shipment
Fact type: ACTOR_ROLE.is_an_initiator_of.TRANSACTION_KIND

	Represented in	Connector point 1	Connector point 2
A09-stock controller.is an initiator of.T08-book shipment-symbol 01	ATD3	Connector point 014	Connector point 015

Figure 78: «A09-stock controller.is an initiator of.T08-book shipment» wiki page

There is a query to obtain all symbol pages that can represent it, because this symbol can be used in several different diagrams. In this case there is only one symbol page, that has two connector points («connector_point_014», «connector_point_015») in diagram ATD3.

«A09-stock controller.is an initiator of.T09-stock control-symbol 01» is also a symbol page represented by a line with two connection points. The red circle in figure 79 shows the symbol placed in the diagram.

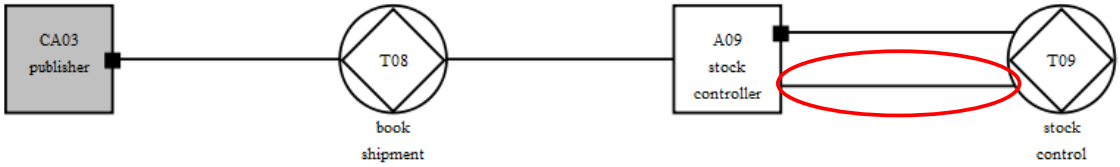

Figure 79: «A09-stock controller.is an initiator of.T09-stock control» symbol

«A09-stock controller.is the executor of.T09-stock control-symbol 01» is also a symbol page represented by a line with two connection points. But this symbol is an instance of «ACTOR_ROLE.is_the_executor_of.TRANSACTION_KIND_SYMBOL». The representation of this symbol is made by adding a black square over one end of the line. The red circle in figure 80 shows the symbol placed in the diagram.

Figure 80: «A09-stock controller.is the executor of.T09-stock control» symbol

«CA03-publisher.is the executor of.T08-book shipment-symbol 01» is a symbol page which is also an instance of «ACTOR_ROLE.is_the_executor_of.TRANSACTION_KIND_SYMBOL». The red circle in figure 81 shows the symbol placed in the diagram.

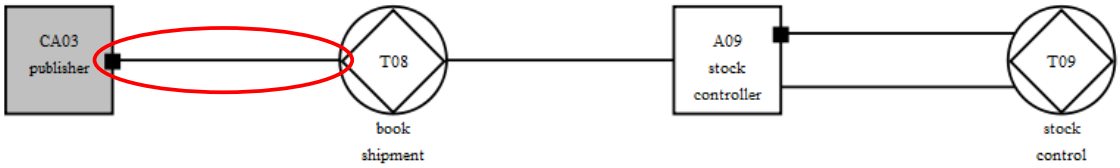

Figure 81: «CA03-publisher.is the executor of.T08-book shipment» symbol

Either we are dealing with instances of «ACTOR_ROLE.is_an_initiator_of.TRANSACTION_KIND_SYMBOL» or «ACTOR_ROLE.is_the_executor_of.TRANSACTION_KIND_SYMBOL» the symbols

are always drawn based on connection points, each one with two coordinates x and y. As a standard, on «ACTOR_ROLE.is_the_executor_of.TRANSACTION_KIND_SYMBOL» instances, the black square is placed over the «connector_point_2» (the ending line point).

«CA03-publisher-symbol 01», «T08-book shipment-symbol 01», and «T09-stock control-symbol 01» are similar to ««A09-stock controller-symbol 01» symbol page.

«CA03-publisher-symbol 01» symbol page represents a composite actor role named «publisher». Figure 82 shows the edit mode for this page where we can see the properties and values defined.

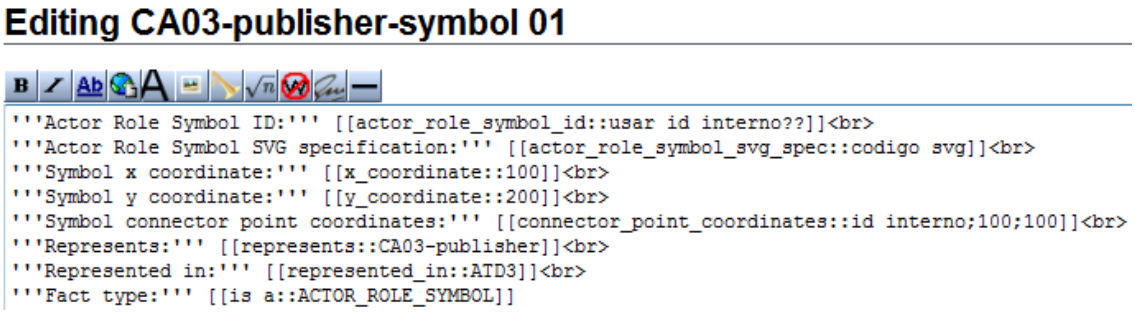

Figure 82: «CA03-publisher-symbol 01» symbol page

SemanticDEMO extension uses only the properties listed in table 8:

Property	Value	Description
x_coordinate	100	Draws the symbol in x coordinate.
y_coordinate	200	Draws the symbol in y coordinate.
represents	CA03-publisher	Represents composite actor role named «publisher»; there is a wiki page to represent it.
represented in	ATD3	Draws the symbol in ATD3 diagram; there is a wiki page to represent it.
is a	ACTOR_ROLE_SYMBOL	This symbol is an instance of ACTOR_ROLE_SYMBOL

Table 8: «CA03-publisher-symbol 01» properties

The red circle in figure 83 shows the symbol placed in the diagram.

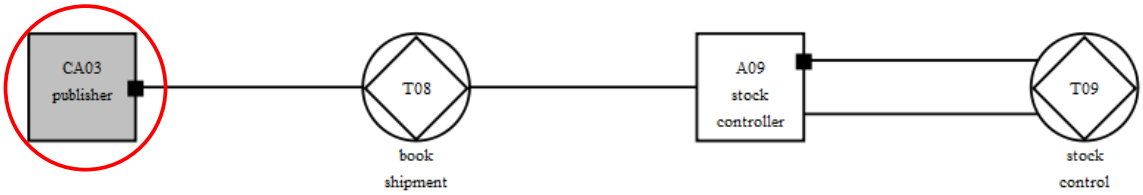

Figure 83: «CA03-publisher» symbol

«T08-book shipment-symbol 01» and «T09-stock control-symbol 01» symbol pages represent two distinct symbols, but they both are instances of «TRANSACTION_KIND_SYMBOL» wiki page.

Thus, let's analyze one symbol page, for example, «T08-book shipment-symbol 01». Figure 84 shows the edit mode for this page where we can see the properties and values defined.

Editing T08-book shipment-symbol 01

```
'''Transaction Kind Symbol ID:''' [[transaction_kind_symbol_id::usar id interno??]]<br>
'''Transaction Kind Symbol SVG specification:''' [[transaction_kind_symbol_svg_spec::codigo svg]]<br>
'''Symbol x coordinate:''' [[x_coordinate::350]]<br>
'''Symbol y coordinate:''' [[y_coordinate::215]]<br>
'''Symbol connector point coordinates:''' [[connector_point_coordinates::id interno;100;100]]<br>
'''Represents:''' [[represents::T08-book shipment]]<br>
'''Represented in:''' [[represented_in::ATD3]]<br>
'''Fact type:''' [[is a::TRANSACTION_KIND_SYMBOL]]
```

Figure 84: «T08-book shipment-symbol 01» symbol page

At this point, *SemanticDEMO* extension uses only the properties listed in table 9:

Property	Value	Description
x_coordinate	350	Draws the symbol in x coordinate.
y_coordinate	215	Draws the symbol in y coordinate.
represents	T08-book shipment	Represents transaction named «book shipment»; there is a wiki page to represent it.
represented in	ATD3	Draws the symbol in ATD3 diagram; there is a wiki page to represent it.
is a	TRANSACTION_KIND_SYMBOL	This symbol is an instance of TRANSACTION_KIND_SYMBOL

Table 9: «T08-book shipment-symbol 01» properties

«T08-book shipment» is a wiki page that represents a transaction named «book shipment». Figure 85 shows the properties and values assigned to this page.

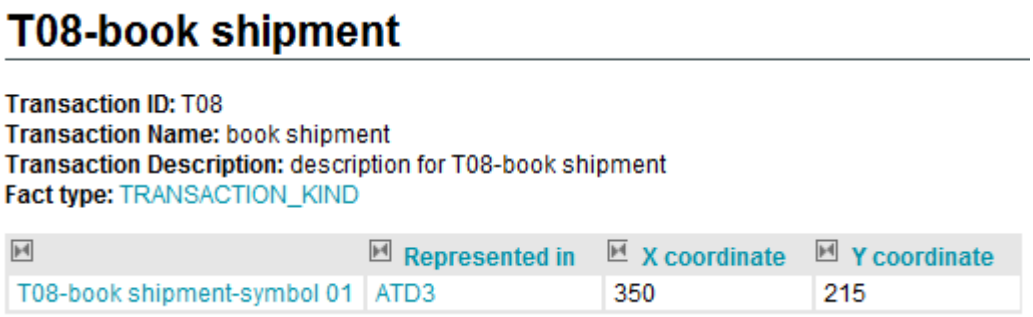

Figure 85: «T08-book shipment» wiki page

We've defined «T08-book shipment» as a «TRANSACTION_KIND» instance, and we've built a query to obtain all symbol pages that can represent it, because this transaction can be used in several different diagrams. In this case there is only one symbol page, that will draw the symbol at coordinates (350, 215) in diagram ATD3. The red circle in figure 86 shows the symbol placed in the diagram.

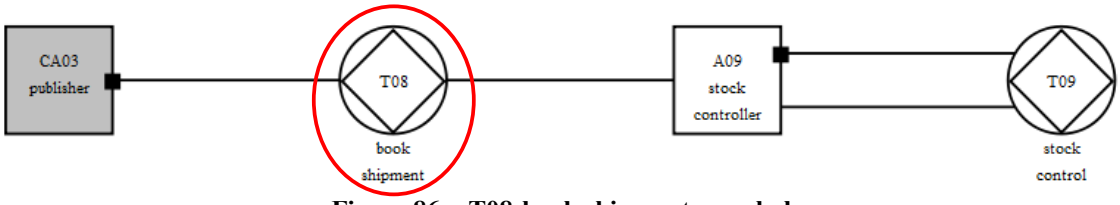

Figure 86: «T08-book shipment» symbol

The generated diagram is built using a dummy svg file. We've embed some functions to allow moving the diagram objects. This is an experimental stage: we can drag and drop the objects, but their coordinates are not updated and line orientation cannot be changed. These features should be implemented in a future work.

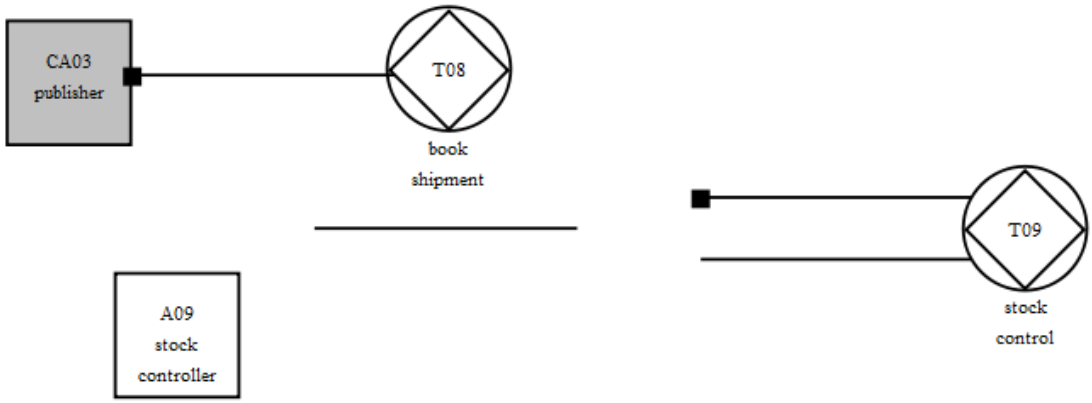

Figure 87: Moving diagram objects

5.7 Problems found

Our first problem arise when we decided to install XAMPP latest release. The problem wasn't related to the package itself but concerned PHP, and MySQL versions that shipped with XAMPP version 1.7.4.

PHP 5.3.5 and MySQL 5.5.8 are not supported by latest *MediaWiki* and *Semantic MediaWiki* versions. So we decided to use XAMPP 1.7.1 that comes with PHP 5.2.9, and MySQL 5.1.33.

But this brings up other issue: the latest *MediaWiki* and *Semantic MediaWiki* versions are not full compatible with PHP 5.2.9, and MySQL 5.1.33. So we decided to use earlier versions: *MediaWiki* 1.15.3 and *Semantic MediaWiki* 1.5.1.

Even if we used the latest versions of MW and SMW we still had compatibility problems with *Halo* extension. The best way to install *Halo* is using the same SMW version. So we installed *Halo* version 1.5.1. Also a great percentage of MW and SMW extensions were developed for stable releases of MW and SWM.

Thus, we found a compromise between stability and version compatibility strictly necessary to overcome our project challenges.

Our second problem was concerned with image shapes path when we were generating a SVG file based on a DOT file. All <image xlink:href» XML tag were missing from the SVG file after running the DOT command-line inside PHP code. Within DOT files we can define image shapes using a «shape» parameter where we have to declare the path of each image. The problem was we weren't declaring the full image path. This was strange because if we write only the image file name and manually run the DOT command-line all turned well. After some testing we managed to understand the problem and started defining full paths.

This leads us to our third problem: the path was a physical path (for example, «C:\xampp\htdocs\wiki\images\dot\someimage.svg») and when presenting a SVG file using a web browser and specially through Apache web server it doesn't work. We managed to overcome this problem by modifying the path to the URL format (for example, «http://localhost/wiki/images/dot/someimage.svg»).

Finally the SVG files were not being presented properly. If we used PNG image shapes all the browsers showed the graph, but if using SVG image shapes Firefox presented blank shapes. The reason for using scalable vector graphics was to be able to scale the vector image and at the same time preserve the shapes. We decided to try other browsers besides Firefox. Opera and Google Chrome managed to correctly present a SVG file where we can zoom in and out without loosing image resolution. But now *Halo* extension doesn't work. *Halo* official website says that some incompatibilities may occur when using other browsers rather than Firefox. Thus this is still an issue in our project, and for know the solution is to use Firefox to work with Halo and use Opera or Chrome to visualize SVG files.

As continuing with our project to developing *SemanticDEMO* extension, we decided to use only Firefox, because it presented the best results. This extension needs to open a

dummy SVG file, write in it, and then save the file with a new name. Due the fact that we already used DOM to achieve this when we modified *SemanticGraph* extension, the procedure was much more easy this time.

6 Future work

There are still some things that can be made to improve the *Semantic MediaWiki* to model organizations.

6.1 Related with *Semantic MediaWiki*

When we create a symbol page, containing the semantic information about a certain symbol, the wiki database isn't updated properly. If we generate a diagram that should contain that symbol, it will not draw the symbol until we edit the page that represents the correspondent organizational artifact and save it. Auto-updating the database should be a future improvement in our project.

Another useful functionality should be auto-renaming a string. For example, if a page title changes, all semantic references to that page should be updated automatically in the database. This is very important because we don't want to manually update 20 or 30 strings.

6.2 Related with *SemanticDEMO* extension

At this point our extension generates a readable diagram and allow the user to move the symbols. But what if we want to save the symbols new position, or dynamically create new connections between symbols? A future improvement will be creating a WYSIWYG editor to allow saving new symbol coordinates and create/remove/rename symbols and connections.

6.3 Related with models

Our project was developed based on ATD diagrams. Thus our extension only generates ATD diagrams. Allowing other diagram types and creating a logic-model independence should be a future improvement in our project.

7 Conclusion

This final chapter is to summarize and remind the important conclusions of our project. We review the project context, problems found, applications used, related work, solutions found and draw the correspondent conclusions.

Our first conclusion is that semantic web is extremely important for creating relations between entities forming a semantic web. We took advantage of semantic web technology to model organizations and used *MediaWiki* combined with *Semantic MediaWiki* extension. Thus, we were able to create wiki pages containing semantic properties, capable of storing the factual information about an organization. This allowed us to obtain a more accurate search for information and better navigation through data.

Our second conclusion is that using existing tools, such as *Graphviz* and *SemanticGraph* extension, isn't the best solution for representing *DEMO* diagrams. We have extended SMW with a modified *SemanticGraph* extension, and although the logical result was satisfactory (the relations between entities were well defined), the design result was not satisfactory (the diagrams become quite confusing and complex when using many resource pages). So we've researched for tools similar to what we intended, and found *Open-Modeling*.

Our third conclusion is that, after the analysis made to *Open-Modeling*, there is no way to integrate both SWM and this tool. *Open-Modeling* runs over Tomcat and it's written in Java, and although we've tried to reuse some code, this showed an impossible task.

Due to the limitations found during our project we've decided to create our own SMW extension to generate *DEMO* diagrams. This extension was based on *SemanticGraph* extension, and we named it *SemanticDEMO*. Basically our extension queries for all wiki pages containing that defined diagram using the property «diagram». The semantic relations are achieved through the «Represented in» property. Thus, the *SemanticDEMO* extension gets all wiki pages that are «represented in» a certain diagram and processes it's contents to automatically generate a SVG file containing the diagram.

We can conclude that by using SMW and our *SemanticDEMO* extension, we offer a secure and reliable way to store and manage semantic relations and its properties, and

also offer a graphical view much more pleasant and understandable.

Our final conclusion is that our project solution represents a big step in supporting *DEMO* methodology because now the organizational reality can be translated into graphical symbols that are critical elements in diagrams, achieving a better understanding of organizational knowledge.

Bibliography

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

[2] "Enterprise Engineering Institute," *DEMO: Enterprise & Engineering Methodology for Organizations*, Jan. 2011.

[3] 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.

[4] A. Ferreira, "Organizational Modeling with a Semantic Wiki: Formalization of content and automatic diagram generation," Universidade da Madeira, 2008.

[5] "MediaWiki."

[6] "Semantic MediaWiki."

[7] Semanticweb.org, "Halo Extension."

[8] "Semantic Graph extension."

[9] "Semantic Forms extension."

[10] "Graphviz - Graph Visualization Software."

[11] "Graphviz."

[12] E. Gansner, E. Koutsofios, e S. North, "Drawing graphs with dot," Dez. 2009.

[13] "Scalable Vector Graphics."

[14] W3C, "Scalable Vector Graphics (SVG) 1.1 (Second Edition)," *W3C Working Draft 22 June 2010*.

[15] "Open-Modeling."

[16] "Inferencing categories and properties."

[17] "Properties and types."

[18] "Technical standard."

[19] "Inline queries."

[20] "Type Record (n-ary properties)."

Annexes

A.1 – Installation manual

The OS used for this installation was Windows Vista, but the install steps should be the same for Windows XP and Windows 7. 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");
```

Figure 88: «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 GNU-Path: «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.2 – Project files

The disc delivered with this document contains all files used in our project. These include the files used for the «MediaWiki» and «Semantic MediaWiki» installation; files modified and hard-coded using PHP; also all the extensions installed.

The purposes of burning all the files in a disc are:

- Allow further improvements to our project, by installing the same versions of all software used;
- Keep this document shorter, instead of pasting here all PHP code.

A.3 – Developed PHP code

For our project we have adapted «SemanticGraph» extension, and created a new extension named «SemanticDEMO».

«SemanticGraph» extension

The following PHP code was developed for «SemanticGraph2.php» file.

```
<?php

/**
 * Semantic Graph Extension - this extension is an
provides some parser functions
 * to create and display graphs based on the structure of
a wiki.
 *
 * To activate this extension, add the following into your
LocalSettings.php file:
 *
require_once('$IP/extensions/SemanticGraph/includes/Semant
icGraph2.php');
 *
 * @ingroup Extensions
 * @author Rob Challen <rjchallen@gmail.com>
 * @version 0.8.5
 * @link
http://www.mediawiki.org/wiki/Extension:MyExtension
Documentation
 * @license http://www.gnu.org/copyleft/gpl.html GNU
General Public License 2.0 or later
 *
 *
 * Changes by Jorge Capela (jorgecapela@hotmail.com)
 * University of Madeira
 * This extension is used for creating DEMO graphs
 * Master thesis oriented by David Aveiro
 */

/**
 * Protect against register_globals vulnerabilities.
 * This line must be present before any global variable is
referenced.
 */
if( !defined( 'MEDIAWIKI' ) ) {
        echo( "This is an extension to the MediaWiki
package and cannot be run standalone.\n" );
        die( -1 );
}
```

```
# Define a setup function
$wgExtensionFunctions[] = 'efSGraphParserFunction_Setup';
# Add a hook to initialise the magic word
$wgHooks['LanguageGetMagic'][]       =
'efSGraphParserFunction_Magic';
// Extension credits that will show up on Special:Version
$wgExtensionCredits['parserhook'][] = array(
        'name'         => 'SemanticGraph',
        'version'      => '0.8.5',
        'author'       => 'Rob Challen',
        'url'          =>
'http://semanticgraph.sourceforge.net/',
        'description'  => 'This extension depends on
graphviz, freemind applet and hypergraph applet'
);

include_once('SemanticGraphSettings.php');
include_once('SemanticGraphFiles.php');
include_once('SemanticGraphRenderer.php');
include_once('SemanticGraphBuilders.php');
include_once('SemanticGraphQuery.php');


// Removed because these functions are not used
//include_once('SemanticGraphHelperFunctions.php');


$wgSemanticGraphSettings = new SemanticGraphSettings();

function efSGraphParserFunction_Setup() {
        global $wgParser;
        # Set a function hook associating the "example"
magic word with our function
        $wgParser->setFunctionHook( 'sgraph',
'efSGraphParserFunction_Render' );
}

function efSGraphParserFunction_Magic( &$magicWords,
$langCode ) {
        # Add the magic word
        # The first array element is case sensitive, in
this case it is not case sensitive
        # All remaining elements are synonyms for our
parser function
        $magicWords['sgraph'] = array( 0, 'sgraph');
        # unless we return true, other parser functions
extensions won't get loaded.
        return true;
}

function efArgs($params,$type) {
      global $wgScriptPath, $wgOut;
      global $wgSemanticGraphSettings;
      if (($args = $wgSemanticGraphSettings-
>parseOptions($params,$type)) == false) {
            $t = '<p>'.$wgSemanticGraphSettings-
>lastError.'</p>';
            $t .= $wgSemanticGraphSettings->usage($type);
```

```
            $wgSemanticGraphSettings->lastError = $t;
            return false;
      }
      return $args;
}

function efSGraphParserFunction_Render( &$parser) {
      global $wgSemanticGraphSettings;
      $initargs = efArgs(func_get_args(),'sgraph');
      if ($initargs == false) return
$wgSemanticGraphSettings->lastError;

      $renderer = new renderer('dot');
      $file = new graphfile($initargs['name'], 'dot');
      $ng = new networkgraph($initargs);

      $dottext = $renderer->enter('preamble', $initargs);
      $dottext .= $initargs['dotoptions'];
      $ng->buildfromWiki();
      $dottext .= $ng->getSubGraph($renderer);
      $dottext .= $renderer->enter('conclusion',
$initargs);


      // Lets do some changes to the dot,
      // before sending the array for creating the file

      //com o explode, criamos um array com todas as
linhas, obtidas pelo separador ;
      $textodividido = explode(";", $dottext);

      foreach ($textodividido as $itemdividido) {
            if ((preg_match("/.is_an_initiator_of./i",
$itemdividido)) && (preg_match("/url/i", $itemdividido)))
            {
                  //echo "ENCONTROU initiator";
                  //echo $itemdividido;
            }
            else {
                  //echo "nao encontrou initiator";
                  $new_no_initiator[] = $itemdividido;
            }
      }



      $novo_sem_initiator = implode (";",
$new_no_initiator);
      //echo $novo_sem_initiator;



      $textodividido2 = explode(";", $novo_sem_initiator);

      foreach ($textodividido2 as $itemdividido2) {
                  if
((preg_match("/.is_the_executor_of./i", $itemdividido2))
&& (preg_match("/url/i", $itemdividido2)))
```

```
                  {
                        //echo "ENCONTROU executor";
                        //echo $itemdividido;
                  }
                  else {
                        //echo "nao encontrou executor";
                        $new_no_executor[] =
$itemdividido2;
                  }
      }


      $novo_sem_executor = implode (";",
$new_no_executor);
      //echo $novo_sem_executor;




      $textodividido3 = explode(";", $novo_sem_executor);

            foreach ($textodividido3 as $itemdividido3) {
                        if ((preg_match("/label=\"T/i",
$itemdividido3)))
                        {
                              //echo "ENCONTROU
Transaction";

                              //tem de copiar a string com
o nome da transaccao
                              //colocar essa string no
tooltip
                              //e elimnar o local original
onde se encontrava
                              //adiciona também o
shapefile

                              $obtem_posicao_transaction =
strpos($itemdividido3, "[label=\"T");

      $obtem_fim_posicao_transaction =
strpos($itemdividido3, "\", URL=");

                              //echo
$obtem_posicao_transaction;

                              $comprimento_string =
$obtem_fim_posicao_transaction -
($obtem_posicao_transaction + 8);

                              $guarda_quase_final =
substr($itemdividido3, ($obtem_posicao_transaction + 8),
$comprimento_string);

                              $posicao_hifen =
strpos($guarda_quase_final, "-") + 1;

                              //echo
```

```
substr($guarda_quase_final, $posicao_hifen,
strlen($guarda_quase_final));

                                $tooltip_final =
substr($guarda_quase_final, $posicao_hifen,
strlen($guarda_quase_final));


                                //$itemdividido3 =
preg_replace("/]/", ",
image=\"C:\\xampp\\htdocs\\wiki\\images\\dot\\img_transact
ion.svg\", tooltip=\"$tooltip_final\"]", $itemdividido3);
                                $itemdividido3 =
preg_replace("/]/", ",
shapefile=\"C:\\xampp\\htdocs\\wiki\\images\\dot\\img_tran
saction.svg\", tooltip=\"$tooltip_final\"]",
$itemdividido3);

                                //echo $itemdividido3;

                                $posicao_correcta =
$obtem_posicao_transaction + $posicao_hifen + 7; //retira
1 porque adicionou 1 em cima (8-1=7)


                                //remove a string do local
(adiciona 1 porque retirou 1 em cima ao $posicao_hifen
                                $novoitem3 =
substr_replace($itemdividido3, '', $posicao_correcta,
($comprimento_string - $posicao_hifen + 1));

                                //echo $novoitem3;


                                $new_good_transaction[] =
$novoitem3;

                        }
                        else {
                                //echo "nao encontrou
transaction";
                                $new_good_transaction[] =
$itemdividido3;
                        }

                }


            $novo_bom_transaction = implode (";",
$new_good_transaction);
            //echo $novo_bom_transaction;




            // nos ACTOR ROLE introduz o shapefile
            // mas no label tem de cortar o texto, ficando
só A... ou CA...
```

```
            // o restante texto vai para tooltip
            $textodividido4 = explode(";",
$novo_bom_transaction);

            foreach ($textodividido4 as $itemdividido4) {



                        if ((preg_match("/label=\"A/i",
$itemdividido4)))
                                    {
                                          //echo
"ENCONTROU elementary actor role";

      $posicao_hifen_actor_role = strpos($itemdividido4,
'-');


      $posicao_antes_label = strpos($itemdividido4,
"[label=\"A");

                                          //echo
$posicao_antes_label;

                                          //obtem o texto
para a tooltip

      $restante_actor_role = substr($itemdividido4,
($posicao_hifen_actor_role+1), ($posicao_antes_label-6));


      $comprimento_actor_role =
strlen($restante_actor_role);


      $dividir_restante_actor_role = explode("_",
$restante_actor_role);

      $juntar_restante_actor_role = implode (" ",
$dividir_restante_actor_role);


      $tooltip_actor_role = $juntar_restante_actor_role;
                                          //
$tooltip_actor_role = ltrim(substr($itemdividido4, 0,
($posicao_antes_label-1)));



                                          //corta a string
para ficar só A...

                                          //calcula
quantos caracteres estao entre A e o hifen

      $diferenca_do_hifen = $posicao_hifen_actor_role - 1;
```

```
                                                $itemdividido4 =
substr_replace($itemdividido4, '',
($posicao_antes_label+8+$diferenca_do_hifen),
($comprimento_actor_role+1));



                                                //$itemdividido4
= preg_replace("/]/", ",
image=\"C:\\xampp\\htdocs\\wiki\\images\\dot\\img_elementa
ry_actor_role.svg\", tooltip=\"$tooltip_actor_role\"]",
$itemdividido4);
                                                $itemdividido4 =
preg_replace("/]/", ",
shapefile=\"C:\\xampp\\htdocs\\wiki\\images\\dot\\img_elem
entary_actor_role.svg\",
tooltip=\"$tooltip_actor_role\"]", $itemdividido4);

                                            }
                        else {
                                                //echo "nao
encontrou elementary actor role";
                        }

                        if ((preg_match("/label=\"CA/i",
$itemdividido4)))
                                            {
                                                //echo
"ENCONTROU composite actor role";


      $posicao_hifen_actor_role = strpos($itemdividido4,
'-');


      $posicao_antes_label = strpos($itemdividido4,
"[label=\"CA");

                                                //echo
$posicao_antes_label;

                                                //obtem o texto
para a tooltip

      $restante_actor_role = substr($itemdividido4,
($posicao_hifen_actor_role+1), ($posicao_antes_label-7));


      $comprimento_actor_role =
strlen($restante_actor_role);


      $dividir_restante_actor_role = explode("_",
$restante_actor_role);

      $juntar_restante_actor_role = implode (" ",
$dividir_restante_actor_role);
```

```
      $tooltip_actor_role = $juntar_restante_actor_role;
                                                //
$tooltip_actor_role = ltrim(substr($itemdividido4, 0,
($posicao_antes_label-1)));



                                                //corta a string
para ficar só CA...

                                                //calcula
quantos caracteres estao entre A e o hifen

      $diferenca_do_hifen = $posicao_hifen_actor_role - 1;


                                                $itemdividido4 =
substr_replace($itemdividido4, '',
($posicao_antes_label+8+$diferenca_do_hifen),
($comprimento_actor_role+1));


                                                //
$tooltip_actor_role = "ainda por fazer";

                                                //$itemdividido4
= preg_replace("/]/", ",
image=\"C:\\xampp\\htdocs\\wiki\\images\\dot\\img_composit
e_actor_role.svg\", tooltip=\"$tooltip_actor_role\"]",
$itemdividido4);
                                                $itemdividido4 =
preg_replace("/]/", ",
shapefile=\"C:\\xampp\\htdocs\\wiki\\images\\dot\\img_comp
osite_actor_role.svg\", tooltip=\"$tooltip_actor_role\"]",
$itemdividido4);

                                        }
                        else {
                                                //echo "nao
encontrou composite actor role";
                        }

                        $new_good_actor_role[] =
$itemdividido4;
            }


            $novo_bom_actor_role = implode (";",
$new_good_actor_role);
            //echo $novo_bom_actor_role;
            //exit(0);




            //
```

```
-----------------------------------------------
            // Parte de baixo, ligações do graph
            //
-----------------------------------------------

            $textodividido5 = explode(";",
$novo_bom_actor_role);

            foreach ($textodividido5 as $itemdividido5) {

                  if
((preg_match("/.is_the_executor_of./i", $itemdividido5)))
                        {
                              //echo "ENCONTROU
is_the_executor_of";

                              $obtem_posicao_isexecutor =
strpos($itemdividido5, ".is_the_executor_of.");
                              //echo
$obtem_posicao_isexecutor;


                              $parte_inicial_executor =
ltrim(substr($itemdividido5, 0,
$obtem_posicao_isexecutor));


                              $obtem_posicao_setalink =
strpos($itemdividido5, "->");
                              //echo
$obtem_posicao_setalink;


                              $item_executor_apos_seta =
substr_replace($itemdividido5, '', 0,
($obtem_posicao_setalink + 3));


                              if
(strcmp($parte_inicial_executor, $item_executor_apos_seta)
== 0) {
                              // iguais

                              }
                              else
                                    {
                                          //diferentes



                              //$comprimento_isexecutor =
$obtem_posicao_setalink - $obtem_posicao_isexecutor;

                              //$novoitem5 =
substr_replace($itemdividido5, '',
$obtem_posicao_isexecutor, ($comprimento_isexecutor - 1));
                              //$new_final_executor[] =
$novoitem5 . "\" [dir=\"back\", arrowhead=\"normal\",
```

```
arrowtail=\"box\"]";


                                $novoitem5 =
$parte_inicial_executor . " -> " .
$item_executor_apos_seta;


                                $new_final_executor[] =
$novoitem5 . "\" [dir=\"back\", arrowtail=\"box\"]";

                            }




                            }
                    else {
                                        //echo "nao encontrou
is_the_executor_of";
                                        $new_final_executor[]
= $itemdividido5;
                            }



            }


            $novo_final_executor = implode (";",
$new_final_executor);
            //echo $novo_final_executor;



            $textodividido6 = explode(";",
$novo_final_executor);

            foreach ($textodividido6 as $itemdividido6) {

                    if
((preg_match("/.is_an_initiator_of./i", $itemdividido6)))
                                        {
                                            //echo
"ENCONTROU is_an_initiator_of";


      $obtem_posicao_isinitiator = strpos($itemdividido6,
".is_an_initiator_of.");
                                        //echo
$obtem_posicao_isexecutor;



      $parte_inicial_initiator =
ltrim(substr($itemdividido6, 0,
```

```
$obtem_posicao_isinitiator));
                                                    //echo
$parte_inicial_initiator;


      $obtem_posicao_setalink = strpos($itemdividido6, "-
>");
                                                    //echo
$obtem_posicao_setalink;

                                                    $item_apos_seta
= substr_replace($itemdividido6, '', 0,
($obtem_posicao_setalink + 3));
                                                    //echo
$item_apos_seta;

                                                    if
(strcmp($parte_inicial_initiator, $item_apos_seta) == 0) {
                                                    // iguais

                                                    }
                                                    else
                                                    {
                                                    //diferentes




                                                    //
$comprimento_isinitiator = $obtem_posicao_setalink -
$obtem_posicao_isinitiator;

                                                    //$novoitem6 =
substr_replace($itemdividido6, '',
$obtem_posicao_isinitiator, ($comprimento_isinitiator -
1));
                                                    //
$new_final_initiator[] = $novoitem6 . "\" [dir=\"none\",
arrowhead=\"normal\", arrowtail=\"normal\"]";

                                                    $novoitem6 =
$item_apos_seta . " -> " . $parte_inicial_initiator;



      $new_final_initiator[] = $novoitem6 . "\"
[dir=\"none\"]";


                                                    }




                                        }
                                    else {
```

```
                                                    //echo
"nao encontrou is_an_initiator_of";

      $new_final_initiator[] = $itemdividido6;
                                        }



                        }


                        $novo_final_initiator = implode
(";", $new_final_initiator);
                        //echo $novo_final_initiator;



      //-----------------------------
      // coloca aspas ""

      $textodividido7 = explode(";",
$novo_final_initiator);

      foreach ($textodividido7 as $itemdividido7) {

            if ((preg_match("/ \[label=/i",
$itemdividido7)))
                        {


                              $itemdividido7 =
preg_replace("/ \[label=/i", "\" [label=",
$itemdividido7);

                              //$new_final_aspas[] = '"' .
$itemdividido7;
                              $newitemdividido7 =
ltrim($itemdividido7);
                              $new_final_aspas[] = "\"" .
$newitemdividido7;


                        }
            else {
                              //echo "nao encontrou
is_an_initiator_of";
                              //$new_final_aspas[] =
$itemdividido7;
                  //}



                  if ((preg_match("/ -> /i",
$itemdividido7)))
                                          {
```

```
      $itemdividido7 = preg_replace("/ -> /i", "\" -> \"",
$itemdividido7);

                                                      //
$new_final_aspas[] = '"' . $itemdividido7;

      $newitemdividido7 = ltrim($itemdividido7);

      $new_final_aspas[] = "\"" . $newitemdividido7;

                                                }
                              else {
                                                      //echo
"nao encontrou is_an_initiator_of";

      $new_final_aspas[] = $itemdividido7;
                  }

            }



      }


      //$novo_final_aspas = implode (";",
$new_final_aspas);
      //echo $novo_final_aspas;



      // remove linhas duplicadas e grava num novo
ficheiro
      $new_final_aspas = array_unique($new_final_aspas);

      //$ltrim_new_final_aspas = ltrim($new_final_aspas);
      //$rtrim_new_final_aspas = rtrim($new_final_aspas);

      //$array_final = implode(";\r\n", $new_final_aspas);
      $array_final = implode(";\r\n", $new_final_aspas);

      //echo $array_final;



      //return array($file->renderGraphFromDot($dottext,
$initargs['engine'], $initargs['width'],
$initargs['height'], $initargs['svg'],
$initargs['boxresize'], $initargs['zoom']), 'noparse' =>
true, 'isHTML' => true);

      return array($file->renderGraphFromDot($array_final,
$initargs['engine'], $initargs['width'],
$initargs['height'], $initargs['svg'],
$initargs['boxresize'], $initargs['zoom']), 'noparse' =>
true, 'isHTML' => true);

      //testing:
```

```
      //return "<pre>".$dottext."</pre>";
}

?>
```

The following PHP code was developed for «SemanticGraphFiles.php» file.

```
function renderGraphFromDot( $timelinesrc, $engine,
$width, $height, $svgview = 'false', $boxresize = "none",
$zoom = "tofit" ) {
            //adapted from renderGraphviz function:
http://www.mediawiki.org/wiki/Extension:GraphViz
        global $wgSemanticGraphSettings;
        global $wgServer;

        $src = $this->filename . '.dot';

            $svg = $this->filename . '-tmp.svg';


        $handle = fopen($src, "w");
        fwrite($handle, $timelinesrc);
        fclose($handle);


        if ($engine == 'dot') {
            $cmd = $wgSemanticGraphSettings->dotCommand;
        } elseif ($engine == 'neato') {
            $cmd = $wgSemanticGraphSettings->neatoCommand;
        } elseif ($engine == 'fdp') {
            $cmd = $wgSemanticGraphSettings->fdpCommand;
        } elseif ($engine == 'circo') {
            $cmd = $wgSemanticGraphSettings->circoCommand;
        } elseif ($engine == 'twopi') {
            $cmd = $wgSemanticGraphSettings->twopiCommand;
        }



        // Executes dot command line to create the SVG
file
        $cmdlinesvg = wfEscapeShellArg($cmd).' -T svg
'.wfEscapeShellArg($src).' -o '.wfEscapeShellArg($svg);
        $WshShell = new COM("WScript.Shell");
            $WshShell->Exec($cmdlinesvg);

        //$cmdlinesvg = wfEscapeShellArg($cmd).' -Tsvg -o
'.wfEscapeShellArg($svg).' '.wfEscapeShellArg($src);
        //@$ret = shell_exec($cmdlinesvg);


            // Open the SVG file (this is a XML file)
            // to add/remove some elements
```

```
            //<text text-anchor="middle" x="191" y="15"
font-family="Arial" font-size="8.00">
            //<tspan x="191" dy="15">exam</tspan>
            //<tspan x="191" dy="30">scheduling</tspan>
            //</text>

            //$file_to_load = $this->filename.'-tmp.svg';
            //$file_to_load = $this->filepath.'-tmp.svg';
            $file_to_load =
'C:\\xampp\\htdocs\\wiki\\images\\dot\\'.$this->name.'-
tmp.svg';

            //echo $file_to_load;

            //exit(0);


            // Use DOMDocument
            $dom = new DOMDocument('1.0', 'UTF-8');
            $dom->xmlStandalone = false;
            $dom->load($file_to_load);
            //$dom-
>load('http://www.example.com/file.xml');
            //$dom->formatOutput = true;


            $tagSVG = $dom->getElementsByTagName('svg')-
>item(0);
            // Get the -height- value
            $attribSVG = $tagSVG-
>getAttributeNode('height');

            //echo $attribSVG->value;

            // Expand the image area by 100pt to allow
transaction text description
            $new_SVG_value = $attribSVG->value;
            $new_SVG_value = str_replace('pt', '',
$new_SVG_value);
            $new_SVG_value = $new_SVG_value + 100;
            $new_SVG_value = $new_SVG_value.'pt';
            $tagSVG->setAttributeNS('', 'height',
$new_SVG_value);

            //echo $new_SVG_value;


            // Find all -image- tags to replace the
physical path string
            // The new path should be http://.... for a
correct image display
            $tagName = $dom-
>getElementsByTagName('image');
            $total_image_tags = $tagName->length;
            //echo $total_image_tags;
            //exit(0);

            // Loop through all the tags
```

```
            for ($counter_i = 0; $counter_i <
$total_image_tags; $counter_i++) {

                        $node = $dom-
>getElementsByTagName('image')->item($counter_i);
                        $attribNode = $node-
>getAttributeNode('xlink:href');
                        $val = $attribNode->value;
                        $val =
str_replace('C:\xampp\htdocs\wiki\images\dot\\',
'http://localhost/wiki/images/dot/', $val);
                        $node-
>setAttributeNS('http://www.w3.org/1999/xlink',
'xlink:href', $val);
                    }



            // Get all g tags with class node
            $all_g = $dom->getElementsByTagName('g');
            //$total_g_nodes = $all_g->length;


            foreach ($all_g as $param_g) {
                    $is_node = $param_g ->
getAttribute('class');

                    $tag_a_from_g = $param_g-
>getElementsByTagName('a')->item(0);


                    $this_title = $param_g-
>getElementsByTagName('title')->item(0);
                    $text_title = $this_title->nodeValue;

                    $is_transaction = strstr ($text_title,
"T");
                    $is_elementary_actor = strstr
($text_title, "A");
                    $is_composite_actor = strstr
($text_title, "CA");

                        if ($is_node == "node") {

                            if ($is_transaction == true)
{
                                //echo "e um node";
                                //echo $text_title;

                                $tag_text_param_x =
$param_g->getElementsByTagName('text')->item(0);
                                $x_value =
$tag_text_param_x -> getAttribute('x');

                                $tag_text_param_y =
$param_g->getElementsByTagName('text')->item(0);
                                $y_value =
```

```
$tag_text_param_y -> getAttribute('y');

                                    //echo $x_value;


      $text_transaction_length = strlen($text_title);

      $text_transaction_hifen = strpos($text_title, "-");
                                    $text_transaction =
substr($text_title, ($text_transaction_hifen + 1),
$text_transaction_length);

                                    $text_array_aux =
explode("_", $text_transaction);


      unset($array_divided_transaction);


      $word_count_transaction=0;

                                    foreach
($text_array_aux as $each_word_transaction) {

      $array_divided_transaction[]=$each_word_transaction;
                                    }


      $word_count_transaction =
count($array_divided_transaction);




                                    //$text_array_aux2 =
implode(" ", $text_array_aux);

                                    //$text_transaction =
$text_array_aux2;

                                    if
($word_count_transaction == 2) {


      $text_transaction = $array_divided_transaction[0];

                                          $element = $dom-
>createElement('text', $text_transaction);
                                          $tag_a_from_g-
>appendChild($element);
                                          $element-
>setAttribute('text-anchor', 'middle');
                                          $element-
>setAttribute('x', $x_value);
                                          $element-
>setAttribute('y', $y_value+25);
                                          $element-
>setAttribute('font-family', 'Arial');
```

```
                                        $element-
>setAttribute('font-size', '8.00');


      $text_transaction = $array_divided_transaction[1];

                                        $element = $dom-
>createElement('text', $text_transaction);
                                        $tag_a_from_g-
>appendChild($element);
                                        $element-
>setAttribute('text-anchor', 'middle');
                                        $element-
>setAttribute('x', $x_value);
                                        $element-
>setAttribute('y', $y_value+35);
                                        $element-
>setAttribute('font-family', 'Arial');
                                        $element-
>setAttribute('font-size', '8.00');


                                    }

                                    if
($word_count_transaction == 3) {


      $first_word_length =
strlen($array_divided_transaction[0]);

                                        if
($first_word_length <= 8) {


      $text_transaction = $array_divided_transaction[0].'
'.$array_divided_transaction[1];

                                            $element =
$dom->createElement('text', $text_transaction);

      $tag_a_from_g->appendChild($element);
                                            $element-
>setAttribute('text-anchor', 'middle');
                                            $element-
>setAttribute('x', $x_value);
                                            $element-
>setAttribute('y', $y_value+25);
                                            $element-
>setAttribute('font-family', 'Arial');
                                            $element-
>setAttribute('font-size', '8.00');


      $text_transaction = $array_divided_transaction[2];

                                            $element =
$dom->createElement('text', $text_transaction);
```

```
      $tag_a_from_g->appendChild($element);
                                                $element-
>setAttribute('text-anchor', 'middle');
                                                $element-
>setAttribute('x', $x_value);
                                                $element-
>setAttribute('y', $y_value+35);
                                                $element-
>setAttribute('font-family', 'Arial');
                                                $element-
>setAttribute('font-size', '8.00');
                                            }
                                            else
                                            {

      $text_transaction = $array_divided_transaction[0];

                                                $element =
$dom->createElement('text', $text_transaction);
      $tag_a_from_g->appendChild($element);
                                                $element-
>setAttribute('text-anchor', 'middle');
                                                $element-
>setAttribute('x', $x_value);
                                                $element-
>setAttribute('y', $y_value+25);
                                                $element-
>setAttribute('font-family', 'Arial');
                                                $element-
>setAttribute('font-size', '8.00');


      $text_transaction = $array_divided_transaction[1].'
'.$array_divided_transaction[2];

                                                $element =
$dom->createElement('text', $text_transaction);
      $tag_a_from_g->appendChild($element);
                                                $element-
>setAttribute('text-anchor', 'middle');
                                                $element-
>setAttribute('x', $x_value);
                                                $element-
>setAttribute('y', $y_value+35);
                                                $element-
>setAttribute('font-family', 'Arial');
                                                $element-
>setAttribute('font-size', '8.00');
                                            }
                                        }

                                        if
($word_count_transaction == 4) {
```

```
      $text_transaction = $array_divided_transaction[0].'
'.$array_divided_transaction[1];

                                                $element = $dom-
>createElement('text', $text_transaction);
                                                $tag_a_from_g-
>appendChild($element);
                                                $element-
>setAttribute('text-anchor', 'middle');
                                                $element-
>setAttribute('x', $x_value);
                                                $element-
>setAttribute('y', $y_value+25);
                                                $element-
>setAttribute('font-family', 'Arial');
                                                $element-
>setAttribute('font-size', '8.00');


      $text_transaction = $array_divided_transaction[2].'
'.$array_divided_transaction[3];

                                                $element = $dom-
>createElement('text', $text_transaction);
                                                $tag_a_from_g-
>appendChild($element);
                                                $element-
>setAttribute('text-anchor', 'middle');
                                                $element-
>setAttribute('x', $x_value);
                                                $element-
>setAttribute('y', $y_value+35);
                                                $element-
>setAttribute('font-family', 'Arial');
                                                $element-
>setAttribute('font-size', '8.00');
                                            }









                                    }



                                if (($is_elementary_actor ==
true) || ($is_composite_actor == true)) {

                                            $tag_text_param_x =
$param_g->getElementsByTagName('text')->item(0);
                                            $x_value =
$tag_text_param_x -> getAttribute('x');

                                            $tag_text_param_y =
$param_g->getElementsByTagName('text')->item(0);
```

```
                                        $y_value =
$tag_text_param_y -> getAttribute('y');

                                        $tag_text_value_null =
$param_g->getElementsByTagName('text')->item(0);
                                        $tag_text_value_null-
>nodeValue="";
                                        $tag_text_value_null-
>setAttribute('text-align', 'center');


                                        $text_actor_length =
strlen($text_title);
                                        $text_actor_hifen =
strpos($text_title, "-");
                                        $text_actor =
substr($text_title, ($text_actor_hifen + 1),
$text_actor_length);
                                        $text_actor_name =
substr($text_title, 0, $text_actor_hifen);

                                        $text_array_aux3 =
explode("_", $text_actor);



      unset($array_divided_text);


      $array_divided_text[]=$text_actor_name;

                                        $word_count=1;

                                        foreach
($text_array_aux3 as $each_word) {

      $array_divided_text[]=$each_word;
                                        }

                                        $word_count =
count($array_divided_text);

                                        $tag_text_from_ga =
$tag_a_from_g->getElementsByTagName('text')->item(0);


                                        if ($word_count == 2)
{
                                              $element2 =
$dom->createElement('tspan', $array_divided_text[0]);

      $tag_text_from_ga->appendChild($element2);
                                              //$element2-
>setAttribute('x', $x_value-3);
                                              $element2-
>setAttribute('x', $x_value);
                                              $element2-
>setAttribute('y', $y_value-10);
```

```
                                        $element2 =
$dom->createElement('tspan', $array_divided_text[1]);

      $tag_text_from_ga->appendChild($element2);
                                        //$element2-
>setAttribute('x', $x_value-3);
                                        $element2-
>setAttribute('x', $x_value);
                                        $element2-
>setAttribute('y', $y_value+10);
                                    }


                                    if ($word_count == 3)
{
                                        $element2 =
$dom->createElement('tspan', $array_divided_text[0]);

      $tag_text_from_ga->appendChild($element2);
                                        $element2-
>setAttribute('x', $x_value);
                                        $element2-
>setAttribute('y', $y_value-10);

                                        $element2 =
$dom->createElement('tspan', $array_divided_text[1]);

      $tag_text_from_ga->appendChild($element2);
                                        $element2-
>setAttribute('x', $x_value);
                                        $element2-
>setAttribute('y', $y_value);

                                        $element2 =
$dom->createElement('tspan', $array_divided_text[2]);

      $tag_text_from_ga->appendChild($element2);
                                        $element2-
>setAttribute('x', $x_value);
                                        $element2-
>setAttribute('y', $y_value+10);
                                    }


                                    if ($word_count == 4)
{
                                        $element2 =
$dom->createElement('tspan', $array_divided_text[0]);

      $tag_text_from_ga->appendChild($element2);
                                        $element2-
>setAttribute('x', $x_value);
                                        $element2-
>setAttribute('y', $y_value-10);

                                        $element2 =
$dom->createElement('tspan', $array_divided_text[1]);
```

```
      $tag_text_from_ga->appendChild($element2);
                                                $element2-
>setAttribute('x', $x_value);
                                                $element2-
>setAttribute('y', $y_value);

                                                $element2 =
$dom->createElement('tspan', $array_divided_text[2]);

      $tag_text_from_ga->appendChild($element2);
                                                $element2-
>setAttribute('x', $x_value);
                                                $element2-
>setAttribute('y', $y_value+10);

                                                $element2 =
$dom->createElement('tspan', $array_divided_text[2]);

      $tag_text_from_ga->appendChild($element2);
                                                $element2-
>setAttribute('x', $x_value);
                                                $element2-
>setAttribute('y', $y_value+20);
                                          }

                                    }


                        }


            }



            $nome_ficheiro =
'C:\xampp\htdocs\wiki\images\dot\\'.$this->name.'.svg';

            //echo $nome_ficheiro;

            $dom->save($nome_ficheiro);



            $iframe = '<IFRAME src="'.$this-
>filepath.'.svg" width="'.$width.'px" height="'.
$height.'px" scrolling="yes" frameborder=1>'
            .'[Your user agent does not support frames or
is currently configured not to display frames. However,
you may visit'
            .'<A href="'.$this->filepath.'-final.svg">the
related document.</A></IFRAME>';
            //$iframe .= '<br/><a href="'.$this-
>filepath.'.dot">Click to download dot source...</a>';

            return $iframe;
```

```
    }
```

The following PHP code was developed for «SemanticGraphRenderer.php» file.

```
function prettyText($str) {
            if ($this->renderer == "hypergraph") {
                  $str = str_replace('<','&lt;',$str);
                  $str = str_replace('>','&gt;',$str);
                  $str = str_replace("'",'&apos;',$str);
                  $str = str_replace("\"",'&quot;',$str);
                  $str = preg_replace('|(&)
(\s*\b\w+\b(?!;))|','$1amp;$2',$str);
            }
            if ($this->renderer == "dot") {
                  $newstr = '';
                  foreach (explode(" ",$str) as $word) {
                        $newstr .= $word;


                        // String are separated by empty
space
                        // New line is what we want, but
not here
                        // The new line changes are made
in XML
                        //if ((strlen($newstr) -
strrpos($newstr, '\n'))>10) {
                              //$newstr .= '\n';
                        //} else {
                              $newstr .= " ";
                        //}
                  }
                  $str = trim($newstr);
                  $str = str_replace("\"",'\"',$str);
                  $str = str_replace("\(",'\(',$str);
                  $str = str_replace("\)",'\)',$str);
            }
            return $str;
      }
```

The following PHP code was developed for «SemanticGraphSettings.php» file.

```
function __construct() {
            global $wgServer, $wgScriptPath;
            if ( ! (stristr (PHP_OS, 'WIN' ) === FALSE) )
{
                  //$this->dotCommand =
'C:/xampp/htdocs/wiki/extensions/SemanticGraph/includes/gr
aphviz_files/bin/dot.exe';
                  $this->dotCommand =
'C:/Graphviz2.26.3/bin/dot';
                  //$this->neatoCommand =
'C:/Graphviz2.26.3/bin/dot.exe';
            } else {
```

```
                $this->dotCommand = '/usr/bin/dot';
                $this->neatoCommand = '/usr/bin/neato';
                $this->circoCommand = '/usr/bin/circo';
                $this->fdpCommand = '/usr/bin/fdp';
                $this->twopiCommand = '/usr/bin/twopi';
        }
```

«SemanticDEMO» extension

The following PHP code was developed for «SemanticDEMO.php» file.

```
<?php

/**
 * Semantic DEMO Extension - this extension provides
functions
 * to create and display SVG graphs based on the structure
of a wiki.
 *
 * To activate this extension, add the following into your
LocalSettings.php file:
 *
require_once('$IP/extensions/SemanticDEMO/includes/Semanti
cDEMO.php');
 *
 * @ingroup Extensions
 * @author Jorge Capela <jorgecapela@hotmail.com> | Master
thesis oriented by David Aveiro | University of Madeira
 * @version 0.0.1
 * @link http://.........
 * @license http://www.gnu.org/copyleft/gpl.html GNU
General Public License 2.0 or later
 *
 * This extension is based on the Semantic Graph extension
structure developed by Rob Challen <rjchallen@gmail.com>
 *
 * SOME IMPORTANT NOTES
 * - after creating the symbol page, i.e. «CA03-publisher-
symbol_01», you need to edit «CA03-publisher» wiki page,
and save it
 */

/**
 * Protect against register_globals vulnerabilities.
 * This line must be present before any global variable is
referenced.
 */
if( !defined( 'MEDIAWIKI' ) ) {
        echo( "This is an extension to the MediaWiki
package and cannot be run standalone.\n" );
        die( -1 );
}
```

```
# Define a setup function
$wgExtensionFunctions[] = 'efSDEMOParserFunction_Setup';
# Add a hook to initialise the magic word
$wgHooks['LanguageGetMagic'][]       =
'efSDEMOParserFunction_Magic';
// Extension credits that will show up on Special:Version
$wgExtensionCredits['parserhook'][] = array(
        'name'          => 'SemanticDEMO',
        'version'       => '0.0.1',
        'author'        => 'Jorge Capela',
        'url'           => 'http://...',
        'description'   => 'This extension is able to
create and display SVG graphs based on the structure of a
wiki'
);

include_once('SemanticDEMOSettings.php');
include_once('SemanticDEMOFunctions.php');
//include_once('SemanticDEMOFiles.php');
//include_once('SemanticDEMORenderer.php');
//include_once('SemanticDEMOBuilders.php');
//include_once('SemanticDEMOQuery.php');



$wgSemanticDEMOSettings = new SemanticDEMOSettings();

function efSDEMOParserFunction_Setup() {
        global $wgParser;
        # Set a function hook associating the "example"
magic word with our function
        $wgParser->setFunctionHook( 'sdemo',
'efSDEMOParserFunction_Render' );
}

function efSDEMOParserFunction_Magic( &$magicWords,
$langCode ) {
        # Add the magic word
        # The first array element is case sensitive, in
this case it is not case sensitive
        # All remaining elements are synonyms for our
parser function
        $magicWords['sdemo'] = array( 0, 'sdemo');
        # unless we return true, other parser functions
extensions won't get loaded.
        return true;
}

function efArgs_DEMO($params,$type) {
      global $wgScriptPath, $wgOut;
      global $wgSemanticDEMOSettings;
      if (($args = $wgSemanticDEMOSettings-
>parseOptions($params,$type)) == false) {
            $t = '<p>'.$wgSemanticDEMOSettings-
>lastError.'</p>';
            $t .= $wgSemanticDEMOSettings->usage($type);
            $wgSemanticDEMOSettings->lastError = $t;
            return false;
```

```
	}
	return $args;
}

function efSDEMOParserFunction_Render( &$parser) {
	global $wgSemanticDEMOSettings;
	$initargs = efArgs_DEMO(func_get_args(),'sdemo');
	//echo $initargs;
	if ($initargs == false) return
$wgSemanticDEMOSettings->lastError;


	$diagram_name = $initargs['diagram'];


	//$file_to_load = 'C:\\xampp\\htdocs\\dummy.svg';

	$file_to_load =
getcwd().'\extensions\SemanticDEMO\svg\\dummy.svg';

	// Use DOMDocument
	$dom = new DOMDocument('1.0', 'UTF-8');
	$dom->xmlStandalone = false;
	$dom->load($file_to_load);



	$tagSVG = $dom->getElementsByTagName('svg')-
>item(0);

	$create_title_node = $dom->createElement("title",
$diagram_name);
	$tagSVG->appendChild($create_title_node);



	$g_index = 0;
	//1 to use the save button
	//$g_index = 1;


$con = mysql_connect("localhost","wikiuser","12345");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

mysql_select_db("wikidb", $con);

// aqui faz a query com uma variavel que trás o nome do
diagrama
$result = mysql_query("SELECT page_title from eomw_page
tbl1 INNER JOIN eomw_pagelinks tbl2 ON tbl1.page_id =
tbl2.pl_from WHERE pl_title='" . $diagram_name . "'");


while($row = mysql_fetch_array($result))
	{
```

```
            //echo $row['page_title'];
            // testar com % antes do -symbol
            $result2 = mysql_query("SELECT smw_id,
smw_title FROM eomw_smw_ids WHERE smw_title LIKE '" .
$row['page_title'] . "-symbol%'");

            while($row2 = mysql_fetch_array($result2))
                  {
                        //estas 2 linhas servem para
eliminar os «.is_an_initiator_of.» e
«.is_the_executor_of.»
                        $is_position =
strpos($row2['smw_title'], ".is_");
                        if ($is_position==0){

                        $result3 = mysql_query("SELECT
smw_id FROM eomw_smw_ids WHERE smw_title =
'X_coordinate'");

                        while($row3 =
mysql_fetch_array($result3))
                                    {

                                          $result30 =
mysql_query("SELECT value_xsd FROM eomw_smw_atts2 WHERE
s_id = '" . $row2['smw_id'] . "' and p_id = '" .
$row3['smw_id'] . "'");
                                          $row30 =
mysql_fetch_array($result30);


                                          $result31 =
mysql_query("SELECT smw_id FROM eomw_smw_ids WHERE
smw_title = 'Y_coordinate'");
                                          while($row31 =
mysql_fetch_array($result31))
                                                {
                                                      $result32
= mysql_query("SELECT value_xsd FROM eomw_smw_atts2 WHERE
s_id = '" . $row2['smw_id'] . "' and p_id = '" .
$row31['smw_id'] . "'");
                                                      $row32 =
mysql_fetch_array($result32);



      $result33 = mysql_query("SELECT smw_id FROM
eomw_smw_ids WHERE smw_title = 'Represents'");

      while($row33 = mysql_fetch_array($result33))
                                                            {

      $result34 = mysql_query("SELECT o_id FROM
eomw_smw_rels2 WHERE s_id = '" . $row2['smw_id'] . "' and
p_id = '" . $row33['smw_id'] . "'");

      while($row34 = mysql_fetch_array($result34))
```

```
      {

            $result35 = mysql_query("SELECT smw_title FROM
eomw_smw_ids WHERE smw_id = '" . $row34['o_id'] . "'");

            while($row35 = mysql_fetch_array($result35))

            {


                  $result36 = mysql_query("SELECT page_id
from eomw_page WHERE page_title='" . $row35['smw_title'] .
"'");

                  $row36 = mysql_fetch_array($result36);




                  $result37 = mysql_query("SELECT pl_title
from eomw_pagelinks WHERE pl_from='" . $row36['page_id'] .
"'");

                  while($row37 =
mysql_fetch_array($result37))

                  {

                        if
(($row37['pl_title']=="COMPOSITE_ACTOR_ROLE") ||
($row37['pl_title']=="ELEMENTARY_ACTOR_ROLE") ||
($row37['pl_title']=="TRANSACTION_KIND")){

                        echo $row37['pl_title'];

                        $symbol_type = $row37['pl_title'];



                        //$symbol_type =
check_symbol_type($row37);

                  //}




                                                echo
"<table border='1'>";
                                                echo
"<tr>";
                                                echo
"<th>Symbol page title</th>";
                                                echo
"<th>Symbol page ID</th>";
                                                echo
"<th>X coord - Property SMW ID</th>";
```

```
                                                    echo
"<th>X coord</th>";
                                                    echo
"<th>Y coord - Property SMW ID</th>";
                                                    echo
"<th>Y coord</th>";
                                                    echo
"<th>Represents</th>";
                                                    echo
"<th>which is a</th>";
                                                    echo
"</tr>";
                                                    echo
"<tr>";
                                                    echo
"<td>" . $row2['smw_title'] . "</td>";
                                                    echo
"<td>" . $row2['smw_id'] . "</td>";
                                                    echo
"<td>" . $row3['smw_id'] . "</td>";
                                                    echo
"<td>" . $row30['value_xsd'] . "</td>";
                                                    echo
"<td>" . $row31['smw_id'] . "</td>";
                                                    echo
"<td>" . $row32['value_xsd'] . "</td>";
                                                    echo
"<td>" . $row35['smw_title'] . "</td>";

                                                    echo
"<td>" . $row37['pl_title'] . "</td>";

                                                    echo
"</tr>";
                                                    echo
"</table>";
                                                    }
                                                    }




      if (($symbol_type=="ELEMENTARY_ACTOR_ROLE") ||
($symbol_type=="COMPOSITE_ACTOR_ROLE")){



      $create_new_g_node = $dom->createElement('g');
      //$create_new_g_node->setAttribute("onclick",
"get_button(evt)");

      $tagSVG->appendChild($create_new_g_node);


      $tag_new_g = $dom->getElementsByTagName('g')-
>item($g_index);
```

```
      // DRAWS «ACTOR_ROLE» SYMBOLS
      $create_new_node_inside_g = $dom-
>createElement('rect');


      $create_new_node_inside_g->setAttribute("id",
"symbol_id_" .$row2['smw_id']);
      $create_new_node_inside_g->setAttribute("height",
"80");
      $create_new_node_inside_g->setAttribute("width",
"80");
      $create_new_node_inside_g->setAttribute("y",
$row32['value_xsd']);
      $create_new_node_inside_g->setAttribute("x",
$row30['value_xsd']);
      $create_new_node_inside_g->setAttribute("stroke-
width", "2");
      $create_new_node_inside_g->setAttribute("stroke",
"#000000");
      if ($symbol_type=="ELEMENTARY_ACTOR_ROLE"){
            $create_new_node_inside_g-
>setAttribute("fill", "#FFFFFF");
      }
      if ($symbol_type=="COMPOSITE_ACTOR_ROLE"){
            $create_new_node_inside_g-
>setAttribute("fill", "#C0C0C0");
      }


      $tag_new_g->appendChild($create_new_node_inside_g);



      //Breaks the text
      $hifen_position = strpos($row35['smw_title'], "-");
      $underscore_position = strpos($row35['smw_title'],
"_");

      $text_first_part = substr($row35['smw_title'], 0,
$hifen_position);


      if ($underscore_position!=0){
            $text_second_part_temp =
substr($row35['smw_title'], $hifen_position+1,
strlen($row35['smw_title']));
            $underscore_new_position =
strpos($text_second_part_temp, "_");
            $text_second_part =
substr($text_second_part_temp, 0,
$underscore_new_position);

      }
      else
      {
```

```
            $text_second_part =
substr($row35['smw_title'], $hifen_position+1,
strlen($row35['smw_title']));
      }


      // TEXT INSIDE ACTOR
      $create_new_node_inside_g = $dom-
>createElement('text', $text_first_part);

      $create_new_node_inside_g->setAttribute("xml:space",
"preserve");
      $create_new_node_inside_g->setAttribute("text-
anchor", "middle");
      $create_new_node_inside_g->setAttribute("font-
family", "serif");
      $create_new_node_inside_g->setAttribute("font-size",
"12");
      $create_new_node_inside_g->setAttribute("id",
"text_id_" .$row2['smw_id']);
      $create_new_node_inside_g->setAttribute("y",
$row32['value_xsd']+30);
      $create_new_node_inside_g->setAttribute("x",
$row30['value_xsd']+40);
      $create_new_node_inside_g->setAttribute("stroke-
width", "0");
      $create_new_node_inside_g->setAttribute("stroke",
"#000000");
      $create_new_node_inside_g->setAttribute("fill",
"#000000");

      $tag_new_g->appendChild($create_new_node_inside_g);



      $create_new_node_inside_g = $dom-
>createElement('text', $text_second_part);

      $create_new_node_inside_g->setAttribute("xml:space",
"preserve");
      $create_new_node_inside_g->setAttribute("text-
anchor", "middle");
      $create_new_node_inside_g->setAttribute("font-
family", "serif");
      $create_new_node_inside_g->setAttribute("font-size",
"12");
      $create_new_node_inside_g->setAttribute("id",
"text_id_" .$row2['smw_id']);
      $create_new_node_inside_g->setAttribute("y",
$row32['value_xsd']+50);
      $create_new_node_inside_g->setAttribute("x",
$row30['value_xsd']+40);
      $create_new_node_inside_g->setAttribute("stroke-
width", "0");
      $create_new_node_inside_g->setAttribute("stroke",
"#000000");
      $create_new_node_inside_g->setAttribute("fill",
"#000000");
```

```
      $tag_new_g->appendChild($create_new_node_inside_g);



      if ($underscore_position!=0){
      $text_third_part = substr($row35['smw_title'],
$underscore_position+1, strlen($row35['smw_title']));

      $create_new_node_inside_g = $dom-
>createElement('text', $text_third_part);

      $create_new_node_inside_g->setAttribute("xml:space",
"preserve");
      $create_new_node_inside_g->setAttribute("text-
anchor", "middle");
      $create_new_node_inside_g->setAttribute("font-
family", "serif");
      $create_new_node_inside_g->setAttribute("font-size",
"12");
      $create_new_node_inside_g->setAttribute("id",
"text_id_" .$row2['smw_id']);
      $create_new_node_inside_g->setAttribute("y",
$row32['value_xsd']+70);
      $create_new_node_inside_g->setAttribute("x",
$row30['value_xsd']+40);
      $create_new_node_inside_g->setAttribute("stroke-
width", "0");
      $create_new_node_inside_g->setAttribute("stroke",
"#000000");
      $create_new_node_inside_g->setAttribute("fill",
"#000000");

      $tag_new_g->appendChild($create_new_node_inside_g);

      }



      //<text xml:space="preserve" text-anchor="middle"
font-family="serif" font-size="24" id="svg_3" y="187"
x="271" stroke-width="0" stroke="#000000"
fill="#000000">hjhjhj</text>


      $g_index = $g_index + 1;

      }







      // DRAWS «TRANSACTION_KIND» SYMBOLS
      if (($symbol_type=="TRANSACTION_KIND")){
```

```
      $create_new_g_node = $dom->createElement('g');


      $tagSVG->appendChild($create_new_g_node);


      $tag_new_g = $dom->getElementsByTagName('g')-
>item($g_index);


      $create_new_node_inside_g = $dom-
>createElement('circle');

      $create_new_node_inside_g->setAttribute("id",
"symbol_id_" .$row2['smw_id']);
      $create_new_node_inside_g->setAttribute("r", "40");
      $create_new_node_inside_g->setAttribute("cy",
$row32['value_xsd']+25);
      $create_new_node_inside_g->setAttribute("cx",
$row30['value_xsd']+40);
      $create_new_node_inside_g->setAttribute("stroke-
width", "2");
      $create_new_node_inside_g->setAttribute("stroke",
"#000000");
      $create_new_node_inside_g->setAttribute("fill",
"#FFFFFF");


      $tag_new_g->appendChild($create_new_node_inside_g);


      $create_new_node_inside_g = $dom-
>createElement('rect');

      $create_new_node_inside_g->setAttribute("transform",
"rotate(45, " .($row30['value_xsd']+40). ", " .
($row32['value_xsd']+40). ")");
      $create_new_node_inside_g->setAttribute("id",
"symbol_id_" .$row2['smw_id']);
      $create_new_node_inside_g->setAttribute("height",
"54");
      $create_new_node_inside_g->setAttribute("width",
"54");
      $create_new_node_inside_g->setAttribute("y",
$row32['value_xsd']+3);
      $create_new_node_inside_g->setAttribute("x",
$row30['value_xsd']+3);
      $create_new_node_inside_g->setAttribute("stroke-
width", "2");
      $create_new_node_inside_g->setAttribute("stroke",
"#000000");
      $create_new_node_inside_g->setAttribute("fill",
"#FFFFFF");
```

```
      $tag_new_g->appendChild($create_new_node_inside_g);




      //Breaks the text
      $hifen_position = strpos($row35['smw_title'], "-");
      $underscore_position = strpos($row35['smw_title'],
"_");

      $text_first_part = substr($row35['smw_title'], 0,
$hifen_position);


      if ($underscore_position!=0){
            $text_second_part_temp =
substr($row35['smw_title'], $hifen_position+1,
strlen($row35['smw_title']));
            $underscore_new_position =
strpos($text_second_part_temp, "_");
            $text_second_part =
substr($text_second_part_temp, 0,
$underscore_new_position);

      }
      else
      {
            $text_second_part =
substr($row35['smw_title'], $hifen_position+1,
strlen($row35['smw_title']));
      }


      // TEXT INSIDE TRANSACTION
      $create_new_node_inside_g = $dom-
>createElement('text', $text_first_part);

      $create_new_node_inside_g->setAttribute("xml:space",
"preserve");
      $create_new_node_inside_g->setAttribute("text-
anchor", "middle");
      $create_new_node_inside_g->setAttribute("font-
family", "serif");
      $create_new_node_inside_g->setAttribute("font-size",
"12");
      $create_new_node_inside_g->setAttribute("id",
"text_id_" .$row2['smw_id']);
      $create_new_node_inside_g->setAttribute("y",
$row32['value_xsd']+30);
      $create_new_node_inside_g->setAttribute("x",
$row30['value_xsd']+40);
      $create_new_node_inside_g->setAttribute("stroke-
width", "0");
      $create_new_node_inside_g->setAttribute("stroke",
"#000000");
      $create_new_node_inside_g->setAttribute("fill",
"#000000");
```

```
      $tag_new_g->appendChild($create_new_node_inside_g);


      // TEXT BELOW TRANSACTION
      $create_new_node_inside_g = $dom-
>createElement('text', $text_second_part);

      $create_new_node_inside_g->setAttribute("xml:space",
"preserve");
      $create_new_node_inside_g->setAttribute("text-
anchor", "middle");
      $create_new_node_inside_g->setAttribute("font-
family", "serif");
      $create_new_node_inside_g->setAttribute("font-size",
"12");
      $create_new_node_inside_g->setAttribute("id",
"text_id_" .$row2['smw_id']);
      $create_new_node_inside_g->setAttribute("y",
$row32['value_xsd']+80);
      $create_new_node_inside_g->setAttribute("x",
$row30['value_xsd']+40);
      $create_new_node_inside_g->setAttribute("stroke-
width", "0");
      $create_new_node_inside_g->setAttribute("stroke",
"#000000");
      $create_new_node_inside_g->setAttribute("fill",
"#000000");

      $tag_new_g->appendChild($create_new_node_inside_g);




      if ($underscore_position!=0){
      $text_third_part = substr($row35['smw_title'],
$underscore_position+1, strlen($row35['smw_title']));

      $create_new_node_inside_g = $dom-
>createElement('text', $text_third_part);

      $create_new_node_inside_g->setAttribute("xml:space",
"preserve");
      $create_new_node_inside_g->setAttribute("text-
anchor", "middle");
      $create_new_node_inside_g->setAttribute("font-
family", "serif");
      $create_new_node_inside_g->setAttribute("font-size",
"12");
      $create_new_node_inside_g->setAttribute("id",
"text_id_" .$row2['smw_id']);
      $create_new_node_inside_g->setAttribute("y",
$row32['value_xsd']+100);
      $create_new_node_inside_g->setAttribute("x",
$row30['value_xsd']+40);
      $create_new_node_inside_g->setAttribute("stroke-
width", "0");
      $create_new_node_inside_g->setAttribute("stroke",
```

```
"#000000");
      $create_new_node_inside_g->setAttribute("fill",
"#000000");

      $tag_new_g->appendChild($create_new_node_inside_g);

      }




      $g_index = $g_index + 1;

      }


      ////***************http://www.php.net/manual/en/func
tion.domnode-append-child.php
      ////http://svg-whiz.com/samples.html



      //<g>
  //<title>Layer 1</title>
  //<polyline se:connector="svg_3 svg_2" fill="none"
stroke-width="5" stroke="#000000" points="396.351,227.5
361.276,177.5 326.202,127.5" id="svg_5"/>
  //<polyline se:connector="svg_1 svg_2" fill="none"
stroke-width="5" stroke="#000000" points="114.5,103.181
191,99.0271 267.5,94.8733" id="svg_4"/>
  //<rect id="svg_1" height="62" width="62" y="74" x="50"
stroke-width="5" stroke="#000000" fill="#FF0000"/>
  //<rect id="svg_2" height="64" width="64" y="61" x="270"
stroke-width="5" stroke="#000000" fill="#FF0000"/>
  //<rect id="svg_3" height="61" width="61" y="230"
x="389" stroke-width="5" stroke="#000000" fill="#FF0000"/>
 //</g>
                                                                  }
                                                                  }
                                                                  }









                                                        }

                                              //}

                                    }

                        }
                  }
```

```
	}


	// «ACTOR_ROLE.is_the_executor_of.TRANSACTION_KIND»
and «ACTOR_ROLE.is_an_initiator_of.TRANSACTION_KIND»

$result = mysql_query("SELECT page_title from eomw_page
tbl1 INNER JOIN eomw_pagelinks tbl2 ON tbl1.page_id =
tbl2.pl_from WHERE pl_title='" . $diagram_name . "'");


while($row = mysql_fetch_array($result))
	{
		//echo $row['page_title'];
		// testar com % antes do -symbol
		$result2 = mysql_query("SELECT smw_id,
smw_title FROM eomw_smw_ids WHERE smw_title LIKE '" .
$row['page_title'] . "-symbol%'");

		while($row2 = mysql_fetch_array($result2))
			{
				//estas 2 linhas servem para
eliminar os «.is_an_initiator_of.» e
«.is_the_executor_of.»
				$is_position =
strpos($row2['smw_title'], ".is_");
				if ($is_position!=0){

				$result21 = mysql_query("SELECT
smw_id FROM eomw_smw_ids WHERE smw_title =
'Connector_point_1'");
				$row21 =
mysql_fetch_array($result21);
				//echo $row21['smw_id'];

				$result22 = mysql_query("SELECT
o_id FROM eomw_smw_rels2 WHERE s_id = '" . $row2['smw_id']
. "' and p_id = '" . $row21['smw_id'] . "'");
				$row22 =
mysql_fetch_array($result22);
				//echo $row22['o_id'];


				$result23 = mysql_query("SELECT
smw_id FROM eomw_smw_ids WHERE smw_title =
'X_coordinate'");
				$row23 =
mysql_fetch_array($result23);
				//echo $row23['smw_id'];

				$result24 = mysql_query("SELECT
value_xsd FROM eomw_smw_atts2 WHERE s_id = '" .
$row22['o_id'] . "' and p_id = '" . $row23['smw_id'] .
"'");
				$row24 =
mysql_fetch_array($result24);
				//echo $row24['value_xsd'];
```

```
                        $result25 = mysql_query("SELECT
smw_id FROM eomw_smw_ids WHERE smw_title =
'Y_coordinate'");
                        $row25 =
mysql_fetch_array($result25);
                        //echo $row25['smw_id'];

                        $result26 = mysql_query("SELECT
value_xsd FROM eomw_smw_atts2 WHERE s_id = '" .
$row22['o_id'] . "' and p_id = '" . $row25['smw_id'] .
"'");
                        $row26 =
mysql_fetch_array($result26);
                        //echo $row26['value_xsd'];




                        $result27 = mysql_query("SELECT
smw_id FROM eomw_smw_ids WHERE smw_title =
'Connector_point_2'");
                        $row27 =
mysql_fetch_array($result27);
                        //echo $row27['smw_id'];

                        $result271 = mysql_query("SELECT
o_id FROM eomw_smw_rels2 WHERE s_id = '" . $row2['smw_id']
. "' and p_id = '" . $row27['smw_id'] . "'");
                        $row271 =
mysql_fetch_array($result271);
                        //echo $row271['o_id'];


                        $result272 = mysql_query("SELECT
smw_id FROM eomw_smw_ids WHERE smw_title =
'X_coordinate'");
                        $row272 =
mysql_fetch_array($result272);
                        //echo $row272['smw_id'];

                        $result273 = mysql_query("SELECT
value_xsd FROM eomw_smw_atts2 WHERE s_id = '" .
$row271['o_id'] . "' and p_id = '" . $row272['smw_id'] .
"'");
                        $row273 =
mysql_fetch_array($result273);
                        //echo $row273['value_xsd'];

                        $result274 = mysql_query("SELECT
smw_id FROM eomw_smw_ids WHERE smw_title =
'Y_coordinate'");
                        $row274 =
mysql_fetch_array($result274);
                        //echo $row274['smw_id'];

                        $result275 = mysql_query("SELECT
value_xsd FROM eomw_smw_atts2 WHERE s_id = '" .
```

```
$row271['o_id'] . "' and p_id = '" . $row274['smw_id'] .
"'");
                        $row275 =
mysql_fetch_array($result275);
                        //echo $row275['value_xsd'];




                        //$result3 = mysql_query("SELECT
smw_id FROM eomw_smw_ids WHERE smw_title =
'X_coordinate'");

                        //while($row3 =
mysql_fetch_array($result3))
                              //{

                                    //$result30 =
mysql_query("SELECT value_xsd FROM eomw_smw_atts2 WHERE
s_id = '" . $row3['smw_id'] . "' and p_id = '" .
$row3['smw_id'] . "'");
                                    //$row30 =
mysql_fetch_array($result30);


                                    //$result31 =
mysql_query("SELECT smw_id FROM eomw_smw_ids WHERE
smw_title = 'Y_coordinate'");
                                    //while($row31 =
mysql_fetch_array($result31))
                                          //{
                                                //
$result32 = mysql_query("SELECT value_xsd FROM
eomw_smw_atts2 WHERE s_id = '" . $row3['smw_id'] . "' and
p_id = '" . $row31['smw_id'] . "'");
                                                //$row32 =
mysql_fetch_array($result32);



      $result33 = mysql_query("SELECT smw_id FROM
eomw_smw_ids WHERE smw_title = 'Represents'");

      while($row33 = mysql_fetch_array($result33))
                                                      {

      $result34 = mysql_query("SELECT o_id FROM
eomw_smw_rels2 WHERE s_id = '" . $row2['smw_id'] . "' and
p_id = '" . $row33['smw_id'] . "'");

      while($row34 = mysql_fetch_array($result34))

      {

            $result35 = mysql_query("SELECT smw_title FROM
eomw_smw_ids WHERE smw_id = '" . $row34['o_id'] . "'");

            while($row35 = mysql_fetch_array($result35))
```

```
            {


                    $result36 = mysql_query("SELECT page_id
from eomw_page WHERE page_title='" . $row35['smw_title'] .
"'");

                    $row36 = mysql_fetch_array($result36);



                    $result37 = mysql_query("SELECT pl_title
from eomw_pagelinks WHERE pl_from='" . $row36['page_id'] .
"'");

                    while($row37 =
mysql_fetch_array($result37))

                    {

                            if
(($row37['pl_title']=="ACTOR_ROLE.is_the_executor_of.TRANS
ACTION_KIND") ||
($row37['pl_title']=="ACTOR_ROLE.is_an_initiator_of.TRANSA
CTION_KIND")){

                            echo $row37['pl_title'];

                            $symbol_type2 =
$row37['pl_title'];









                                                    echo
"<table border='1'>";
                                                    echo
"<tr>";
                                                    echo
"<th>Symbol page title</th>";
                                                    echo
"<th>Symbol page ID</th>";
                                                    //echo
"<th>connector_point_1 - Property SMW ID</th>";
                                                    echo
"<th>connector_point_1</th>";
                                                    //echo
"<th>connector_point_2 - Property SMW ID</th>";
                                                    echo
"<th>connector_point_2</th>";
                                                    echo
```

```
"<th>Represents</th>";
                                                    echo
"<th>which is a</th>";
                                                    echo
"</tr>";
                                                    echo
"<tr>";
                                                    echo
"<td>" . $row2['smw_title'] . "</td>";
                                                    echo
"<td>" . $row2['smw_id'] . "</td>";
                                                    //echo
"<td>" . $row3['smw_id'] . "</td>";
                                                    echo
"<td>" . $row24['value_xsd'] . "," . $row26['value_xsd'] .
"</td>";
                                                    //echo
"<td>" . $row31['smw_id'] . "</td>";
                                                    echo
"<td>" . $row273['value_xsd'] . "," . $row275['value_xsd']
. "</td>";
                                                    echo
"<td>" . $row35['smw_title'] . "</td>";

                                                    echo
"<td>" . $row37['pl_title'] . "</td>";

                                                    echo
"</tr>";
                                                    echo
"</table>";
                                                    }
                                                    }




      if
(($symbol_type2=="ACTOR_ROLE.is_the_executor_of.TRANSACTIO
N_KIND") ||
($symbol_type2=="ACTOR_ROLE.is_an_initiator_of.TRANSACTION
_KIND")){


      $create_new_g_node = $dom->createElement('g');


      $tagSVG->appendChild($create_new_g_node);


      $tag_new_g = $dom->getElementsByTagName('g')-
>item($g_index);


      $create_new_node_inside_g = $dom-
>createElement('polyline');
```

```
      $create_new_node_inside_g->setAttribute("id",
"symbol_id_" .$row2['smw_id']. "_polyline");
      $create_new_node_inside_g->setAttribute("points",
$row24['value_xsd'] . " " . $row26['value_xsd'] . " " .
$row273['value_xsd'] . " " . $row275['value_xsd']);
      $create_new_node_inside_g->setAttribute("stroke-
width", "2");
      $create_new_node_inside_g->setAttribute("stroke",
"#000000");
      $create_new_node_inside_g->setAttribute("fill",
"#FFFFFF");


      $tag_new_g->appendChild($create_new_node_inside_g);


      if
($symbol_type2=="ACTOR_ROLE.is_the_executor_of.TRANSACTION
_KIND"){
      $create_new_node_inside_g = $dom-
>createElement('rect');

      $create_new_node_inside_g->setAttribute("id",
"symbol_id_" .$row2['smw_id']. "_rect");
      $create_new_node_inside_g->setAttribute("height",
"10");
      $create_new_node_inside_g->setAttribute("width",
"10");
      $create_new_node_inside_g->setAttribute("y",
$row275['value_xsd']-4);
      $create_new_node_inside_g->setAttribute("x",
$row273['value_xsd']);
      $create_new_node_inside_g->setAttribute("stroke-
width", "2");
      $create_new_node_inside_g->setAttribute("stroke",
"#000000");
      $create_new_node_inside_g->setAttribute("fill",
"#000000");


      $tag_new_g->appendChild($create_new_node_inside_g);
      }


      $g_index = $g_index + 1;

      //<polyline se:connector="svg_3 svg_2" fill="none"
stroke-width="5" stroke="#000000" points="396.351,227.5
361.276,177.5 326.202,127.5" id="svg_5"/>
      }




                                                      }
```

```
                                                                        }
                                                                        }

                                                            //}

                                                      //}

                                                //}

                  }

                  }

            }

mysql_close($con);

//echo getcwd();

//$nome_ficheiro = 'C:\xampp\htdocs\\'.
$initargs['diagram'].'.svg';

$nome_ficheiro =
getcwd().'\extensions\SemanticDEMO\svg\\'.
$diagram_name.'.svg';

//echo $nome_ficheiro;

      //echo $nome_ficheiro;
      $dom->save($nome_ficheiro);

$show_created_svg = "Diagram: ".$diagram_name."<br />";
$show_created_svg = $show_created_svg."View SVG:
http://localhost/wiki/extensions/SemanticDEMO/svg/".
$diagram_name.".svg";
```

```
return $show_created_svg;

//$iframe = "<IFRAME src='http://localhost/new-dummy.svg'
width='800px' height='600px' scrolling='yes'
frameborder='1' />";

//return $iframe;

}


?>
```

The following SVG code was developed for «dummy.svg» file.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="1024pt" height="768pt"
      xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:a3="http://ns.adobe.com/AdobeSVGViewerExtension/3.0/
"
      xmlns="http://www.w3.org/2000/svg"
xmlns:se="http://svg-edit.googlecode.com"
   onload='Init(evt)'
   onmousedown='Grab(evt)'
   onmousemove='Drag(evt)'
   onmouseup='Drop(evt)'>

   //<script type="text/ecmascript"
xlink:href="../includes/helper_functions.js"/>
   //<script type="text/ecmascript"
xlink:href="../includes/timer.js"/>
   //<script type="text/ecmascript"
xlink:href="../includes/button.js"/>

   <script type="text/ecmascript"><![CDATA[
      var SVGDocument = null;
      var SVGRoot = null;

      var TrueCoords = null;
      var GrabPoint = null;
      var BackDrop = null;
      var DragTarget = null;

        //var textbutton1;

      function Init(evt)
      {


            SVGDocument = evt.target.ownerDocument;
        SVGRoot = SVGDocument.documentElement;

         // these svg points hold x and y values...
```

```
         //    very handy, but they do not display on the
screen (just so you know)
         TrueCoords = SVGRoot.createSVGPoint();
         GrabPoint = SVGRoot.createSVGPoint();

         // this will serve as the canvas over which items
are dragged.
         //    having the drag events occur on the
mousemove over a backdrop
         //    (instead of the dragged element) prevents
the dragged element
         //    from being inadvertantly dropped when the
mouse is moved rapidly
         BackDrop =
SVGDocument.getElementById('BackDrop');

      }

      function Grab(evt)
      {
         // find out which element we moused down on
             //var targetElement = evt.target;

             // Grabs all elements inside a <g> tag (by
Jorge Capela)
         var targetElement = evt.target.parentNode;

         // you cannot drag the background itself or the
save button, so ignore any attempts to mouse down on it
         if ( BackDrop != targetElement )
         {
            //set the item moused down on as the element
to be dragged
            DragTarget = targetElement;

            // move this element to the "top" of the
display, so it is (almost)
            //    always over other elements (exception:
in this case, elements that are
            //    "in the folder" (children of the folder
group) with only maintain
            //    hierarchy within that group
            DragTarget.parentNode.appendChild( DragTarget
);

            // turn off all pointer events to the dragged
element, this does 2 things:
            //    1) allows us to drag text elements
without selecting the text
            //    2) allows us to find out where the
dragged element is dropped (see Drop)
            DragTarget.setAttributeNS(null, 'pointer-
events', 'none');

            // we need to find the current position and
translation of the grabbed element,
            //    so that we only apply the differential
between the current location
```

```
            //    and the new location
            var transMatrix = DragTarget.getCTM();
            GrabPoint.x = TrueCoords.x -
Number(transMatrix.e);
            GrabPoint.y = TrueCoords.y -
Number(transMatrix.f);
         }
      };


      function Drag(evt)
      {
         // account for zooming and panning
         GetTrueCoords(evt);

         // if we don't currently have an element in tow,
don't do anything
         if (DragTarget)
         {
            // account for the offset between the
element's origin and the
            //    exact place we grabbed it... this way,
the drag will look more natural
            var newX = TrueCoords.x - GrabPoint.x;
            var newY = TrueCoords.y - GrabPoint.y;

            // apply a new tranform translation to the
dragged element, to display
            //    it in its new location
            DragTarget.setAttributeNS(null, 'transform',
'translate(' + newX + ',' + newY + ')');
         }
      };


      function Drop(evt)
      {
         // if we aren't currently dragging an element,
don't do anything
         if ( DragTarget )
         {
            // since the element currently being dragged
has its pointer-events turned off,
            //    we are afforded the opportunity to find
out the element it's being dropped on
            var targetElement = evt.target;

            // turn the pointer-events back on, so we can
grab this item later
            DragTarget.setAttributeNS(null, 'pointer-
events', 'all');
            if ( 'Folder' == targetElement.parentNode.id )
            {
               // if the dragged element is dropped on an
element that is a child
               //    of the folder group, it is inserted
as a child of that group
               targetElement.parentNode.appendChild( DragT
```

```
arget );
               //alert(DragTarget.id + ' has been dropped
into a folder, and has been inserted as a child of the
containing group.');
            }
            else
            {
               // for this example, you cannot drag an
item out of the folder once it's in there;
               //    however, you could just as easily do
so here
               //alert(DragTarget.id + ' has been dropped
on top of ' + targetElement.id);
            }

            // set the global variable to null, so nothing
will be dragged until we
            //    grab the next element
            DragTarget = null;



             }
      };


      function GetTrueCoords(evt)
      {
         // find the current zoom level and pan setting,
and adjust the reported
         //    mouse position accordingly
         var newScale = SVGRoot.currentScale;
         var translation = SVGRoot.currentTranslate;
         TrueCoords.x = (evt.clientX -
translation.x)/newScale;
         TrueCoords.y = (evt.clientY -
translation.y)/newScale;
      };



   ]]></script>



</svg>
```

A.4 Testing *SemanticGraph* and *SemanticDEMO* extensions

To obtain a diagram using *SemanticGraph* extension:

1. Browse «http://localhost/wiki/index.php/Special:AllPages»
    a) Click on «Teste_SemanticGraph2» page (you must use Opera or Google Chrome browser), and you should see figure 89.

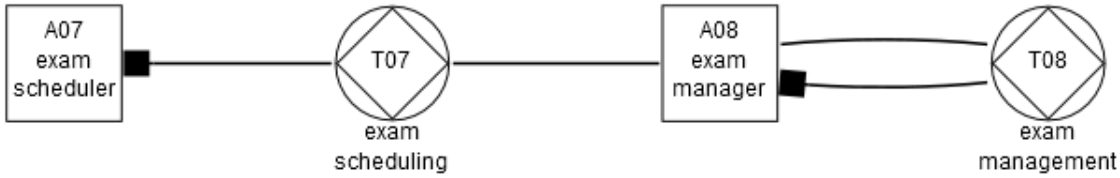

Figure 89: DEMO diagram (ATD) generated using *SemanticGraph* extension

To obtain a diagram using *SemanticDEMO* extension:

1. Browse «http://localhost/wiki/index.php/Special:AllPages»
    a) Click on «ATD3» page, and you should see figure 90.

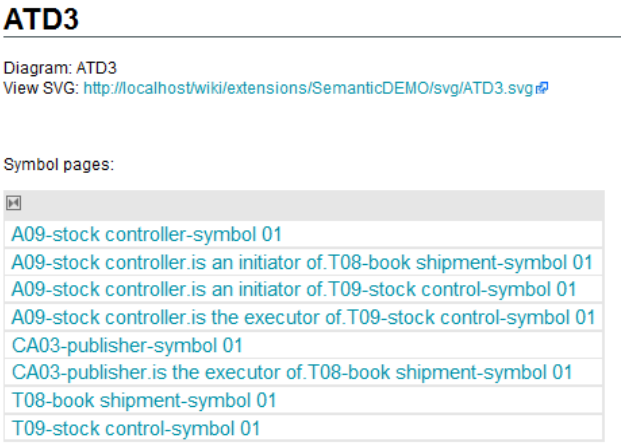

Figure 90: ATD3 test page after generating the diagram using *SemanticDEMO* extension

b) Click on the link next to «View SVG:», and you should see figure 91.

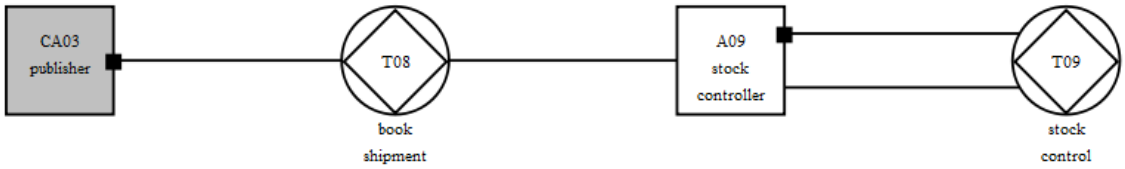

Figure 91: DEMO diagram (ATD) generated using *SemanticDEMO* extension