International Journal of Software Engineering
and Knowledge Engineering
Vol. 14, No. 3 (2004) 323–349

World Scientific
www.worldscientific.com

AN ONTOLOGY FOR THE MANAGEMENT OF SOFTWARE MAINTENANCE PROJECTS

FRANCISCO RUIZ*, AURORA VIZCAÍNO†, MARIO PIATTINI‡ and FELIX GARCÍA§

Alarcos Research Group, Escuela Superior de Informática,
Universidad de Castilla-La Mancha, Paseo de la Universidad 4, 13071 Ciudad Real, Spain
*Francisco.RuizG@uclm.es
†Aurora.Vizcaino@uclm.es
‡Mario.Piattini@uclm.es
§Felix.Garcia@uclm.es

Submitted 10 February 2003
Received 12 December 2003
Accepted 20 December 2003

Different proposals exist to represent the software maintenance process. However most of them are very informal or too focussed on a specific goal. We have developed a semi-formal ontology where the main concepts, according to the literature related to software maintenance, have been described. This ontology, besides representing static aspects, also represents dynamic issues related to the management of software maintenance projects. In order to develop an ontology a suitable methodology should also be followed. REFSENO was the methodology used in this work. The ontology that this work presents is not a preliminary idea but it has already been used in software maintenance environments, such as MANTIS, which is currently working successfully.

Keywords: Software maintenance; software projects management; ontology; REFSENO.

1. Introduction

Many studies [7, 40] have demonstrated that the majority of the overall expenses incurred during the life-cycle of a software product, occur during the maintenance stages. Thus, in recent years, researchers have focussed their attention on looking for techniques which help to increase the efficiency of the Software Maintenance Process (SMP).

One way to improve maintenance quality and decrease maintenance costs is to reuse previous information and knowledge [32]. However, for information to be usable it needs to be modelled, structured, generalised and stored in a reusable form, with the goal of allowing effective retrieval [2].

†Contact author.

In order to decrease the efforts and costs of the SMP we developed MANTIS [45], an "extended software engineering environment" to manage maintenance projects. The ontology presented in this paper is one of the four elements of the MANTIS conceptual framework. The other three are a multilevel conceptual architecture, a processes system, and a collection of metamodels. Now a new component, called the KM module, is being added to MANTIS. This module is in charge of fostering the reuse of all information, knowledge and expertise generated during the SMP. KM-MANTIS is based on the experience factory concept [3, 4] since it is known that an organisational memory must be maintained by an organisational unit, which is separate from the project organisations. This is because it is mainly concerned with keeping to schedules and cost constraints, and providing knowledge would imply extra effort.

Before constructing the systems, modelling, structuring and generalising the information that is generated during SMP is vitally important. In order to reach this goal we decided to construct a common conceptualisation of the domain, where objects, concepts, entities and their relationships were explicitly represented. For this we used ontologies, which enable explicit specification of a conceptualisation [18]. An ontology represents a certain view of an application domain in which the concepts that live in this domain are defined in an unambiguous and explicit way [8].

Moreover, as is explained in [33] ontologies facilitates enterprise knowledge management, "knowledge sharing" [37], and knowledge integration [11]. There are requirements which are very important for KM-MANTIS, since its goal is to promote the sharing and reusing of information and knowledge.

Different authors, such as [8], have stressed the convenience of using ontologies to reuse knowledge in maintenance activities. [8] presents a concept-oriented approach, focusing on the concepts to be reused, which depend on each organisation. Our goal is to define an ontology in a higher level of abstraction, since the ontologies that we present describe the SMP itself.

The only known work in this area has been carried out at [29]'s proposal, where the main aspects to be taken into account in empirical studies of software maintenance were described. However, in order to manage software maintenance projects, besides identifying the factors that influence maintenance as in Kitchenham *et al.*'s ontology it is also necessary to indicate a minimal set of dynamic aspects of the domain, described in terms of states, transitions, events, processes, etc. Thus, we decided to design a more extended and semi-formal ontology, (which, for reasons of clarity, is organized into three ontologies and four subontologies) where both static and dynamic aspects were clearly defined and specified. These last ones are modeled by means of workflows [47]. Moreover, our ontology presents a focus different from that of [29]'s, since our ontology pretends to be more practical and oriented towards managing maintenance projects from a point of view of a business process.

This paper describes the different ontologies and subontologies that are part of the global ontology to manage software maintenance projects. The methodology

used to define the ontology is also explained.

The contents of this paper are organised as follows: Section 2 describes what features and parts, according to the literature and standards studied, would be convenient to consider in order to design an ontology that represents the SMP. Section 3 is where the methodology used to develop the ontology is explained. Later in Sec. 4, the main section, the different ontologies and subontologies, which form the global SMP ontology, are described in detail. Section 5 describe our experience using this ontology in real projects. Finally, conclusions are presented in Sec. 6.

2. An Ontology for Software Maintenance

The process of software maintenance involves different kinds of information which come from different sources (projects, staff, methodologies, etc.). Due to the complexity of the SMP, a huge ontology is required, which integrates aspects related to the four types of ontologies defined by [36] for information systems:

- Static ontologies encompass static aspects of an application, described in terms of concepts such as entity, attribute, relationship, or resources.
- Dynamic ontologies, which describe dynamic aspects within an application, described in terms of states, transitions, events, processes, etc.
- Intentional ontologies describe aspects related to the different agents (human or otherwise).
- Social ontologies describe social settings in terms of social relationships among agents.

Our ontology is made up of a set of ontologies (see Fig. 1), which represent the different features defined by Mylopoulos. In order to represent the static aspects, we defined the **Maintenance Ontology**, which consists of four subontologies. They describe the concepts related to maintenance and consist of a subontology for products, another for activities, a third for processes and the fourth for describing the different agents involved in SMP. The number of static ontologies coincides with those proposed by [29]. Nevertheless we have extended and changed them.

The dynamic part is represented by an ontology called **Workflow Ontology**, where three relevant aspects of the maintenance process are defined:

- Decomposition of activities.
- Temporal constraint between activities. This being the order in which the activities must be performed.
- Control of the execution of activities and projects during the process enactment.

A third ontology called a **Measurement ontology** represents both static and dynamic aspects related to the software measurement. An example of a dynamic aspect is the actions to be measured. This ontology has been included because of the importance of measurement within the software process.

The intentional and social aspects are considered within the same subontology, Agents, since they are closely related.

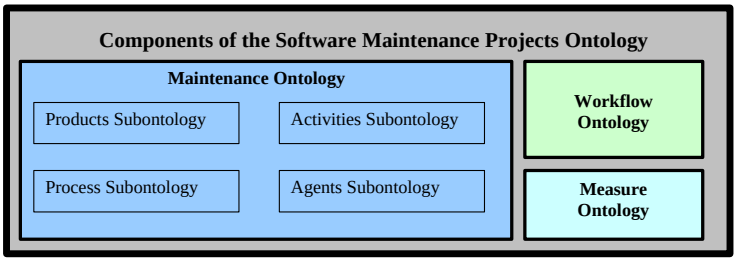

Fig. 1. Structure of the software maintenance projects ontology.

Therefore, our ontology considers the four aspects that should be taken into account when an information system is modelled. This is one of the contributions of this work.

However, considering the four aspects is not enough to ensure a formal ontology. An ontology must be clear and precise, communicating the meaning of its terms in an efficient way. Moreover, it should be coherent, allowing the making of consistent inferences with the definitions. The methodology used to define the ontology helped us to solve some of these issues, such as clarity since the methodology enabled us to use different types of representation of the information and different layers to classify it. Moreover, a method of integration was used to avoid redundancies or inconsistencies. Although there are highly elaborate proposals defining very complete complex processes for the integration of ontologies [30, 41], we have opted for a simpler solution using the iterative method proposed in [35]. This way, the ontology presented in this paper is the result of carrying out the three following steps during two iterations:

1. To search for the items (concepts, attributes or interrelations) where overlapping exists.
2. To relate concepts that are semantically linked by mean of equivalences or relationships of the same class (alignment).
3. To check the consistency, coherence and absence of redundancy of the results.

The different steps followed to construct the ontology are described in the following section.

3. Ontology Engineering Method

Many authors have pointed out the importance of formalising the definition and design of information systems with ontologies [20]. However, to design an ontology it is advisable to follow a methodology suitable for this goal. Different methodologies and representations have been proposed. For instance, [21] uses a representation based on first-order predicate logic. Other authors prefer frame-based approaches, such as those that are used in Ontolingua [12], one of the most frequently used ontologic languages.

We have chosen the REFSENO (*Representation Formalism for Software Engineering Ontologies*) methodology, proposed by Tautz and Von Wangenheim [51]. The main reasons that motivated us to use this methodology were:

1. REFSENO was specifically designed for software engineering.
2. It allows the modelling of software engineering knowledge by using alternate representations. The other approaches [12, 17, 21, 50, 52, 53] only allow types of representation which are less intuitive of understanding for people who are not familiar with first-order predicate logic or equivalent.
3. It has a clear terminology, differentiating between conceptual and context-specific knowledge, thus enabling the management of knowledge from different contexts. On the contrary, the above approaches do not distinguish the context-specific knowledge. Furthermore, they represent a higher level of abstraction. Therefore, they represent lesser level of granularity than REFSENO does.
4. It helps to build consistent ontologies thanks to the use of consistency criteria.
5. The other sets of techniques do not use constructs known from Case-Based Reasoning (CBR) as REFSENO does.
6. It follows the indications of [1] in the sense that the storage of experience in the form of documents (in contrast to codified knowledge) results in an important reduction of learning effort, typically associated with knowledge-based systems.

On the other hand, the epistemistic primitives of REFSENO draw their ideas from several areas closely related to the areas that we use in the implementation of KM-MANTIS. These are areas such as database mechanisms (e.g., relationships between concepts and implied consistency rules), CBR mechanisms (e.g., similarity-based retrieval) and knowledge-based mechanisms (e.g., inference rules). In addition, the representation formalism is object-centred, since it uses a model similar to UML (Unified Modelling Language). We have directly used UML to draw the ontologic diagrams since it is a standard which is well known to the entire software engineering community.

3.1. *Brief REFSENO description*

REFSENO provides constructs to describe concepts where each concept represents a class of experience items. Besides concepts, its properties (called terminal attributes) and relationships (nonterminal attributes) are represented.

One relevant feature of REFSENO is that it enables us to describe similarity functions, which are used for similarity-based retrieval. In this way the methodology facilitates the implementation of the retrieval component. On the other hand, REFSENO also incorporates integrity rules such as: cardinalities and value ranges for attributes, assertions, and preconditions that the instance must fulfil. REFSENO extends the formalism of [39] by additional integrity rules, and by clearly separating the schema definition and characterisation.

REFSENO is an improved adaptation of *Methontology* [13, 17] which imitates

the software life-cycle proposed by the IEEE 1074 standard [22], where the main stages are:

1. Planning.
2. Specification of the ontology requirements.
3. Conceptualisation (similar to the phase of design in a software system, so it is the ontology itself).
4. Implementation (this means the representation and storing of the previous conceptualisation by using computer science tools).

In REFSENO, the detailed information of the ontology is represented by means of a collection of tables: concepts glossary, table of attributes, table of relationships (non terminal attributes), of relationship classes, etc.

Terminal concept attributes are described by a 9-tuple formed from the following items:

- Name: The name is used for reference purposes.
- Description: A narrative text which defines the meaning of the attribute.
- Cardinality: A range specifying the minimum and maximum number of values the attribute may have.
- Type: Each terminal concept attribute is given a type, and the types are viewed as an epistemistic primitive. REFSENO has some predefined types such as Boolean, Integer, Real, Text, Identifier or Date. New types can be described by users.
- Default value: This is related to the insertion of new instances. If the user entering a new instance does not specify a value for this attribute, the default value is used.
- Mandatory: This is also related to new instances. It indicates whether an attribute value of an instance has to be specified.
- Value inference: This component defines how to calculate the attribute value automatically (if possible) based on the values of other attributes.
- Inferred attributes: This component lists all the attributes whose value is inferred using a value of this attribute. There is a mutual dependence between value inferences and inferred attributes, thus inferred attributes can automatically be derived from the value inferences.
- Standard weight: This weight may be used by the similarity functions of the concept this attribute belongs to. A weight of 0 denotes an attribute whose value will not be used for querying.

REFSENO distinguishes three layers to which attributes may belong. These are artifact, interface and context. The attributes of the artifact layer characterise the instances themselves. Attributes of the interface layer characterise how a particular instance can be integrated into the system. Attributes of the context layer characterise the environment in which the instance has been applied and the quality of the instance in the specified environment.

In order to calculate the similarity functions between the two instances i and i' the different layers should be taken into account, since there is a similarity function for each layer. For a concept c these are $\mathrm{sim}_{\mathrm{artif}}(c)$, $\mathrm{sim}_{\mathrm{I/F}}(c)$ and $\mathrm{sim}_{\mathrm{ctxt}}(c)$ and this is based on the local similarity functions of the concept's attributes. The values of similarity functions for a concept c between two instances i and i' are combined to a single similarity value as follows:

$$\mathrm{Sim}(c)(i,i') = W_{\mathrm{artif}} * \mathrm{sim}_{\mathrm{artif}}(c)(i,i') + W_{\mathrm{I/F}} * \mathrm{sim}_{\mathrm{I/F}}(c)(i,i') + W_{\mathrm{ctxt}} * \mathrm{sim}_{\mathrm{ctxt}}(c)(i,i'),$$

where W_{artif}, $W_{\mathrm{I/F}}$, W_{ctxt} are weights with which the similarity functions can be adjusted to the needs of the users. The sum of the weights is always 1. A similarity value equal to 0 means total dissimilarity between i and i', and a value equal to 1 indicates total similarity (equivalence). The concept's similarity functions are of a global nature because they are based on the local similarity functions of the concept's attributes. An example of how a similarity function is calculated will be described in greater length when attributes tables are shown. Besides similarity functions, attributes tables may also have assertions which are conditions expressed as a formula, and that all instances must fulfil.

The nonterminal attributes, those that represent how a particular entity is related to other entities, can be represented in the same concept attribute table used for the terminal concept attributes. REFSENO allows other possible representations for nonterminal attributes, for example graphically, by using a tree structure. However, in this paper only the first representation (tables) is used.

REFSENO has various kinds of predefined relationships, these being*: is-a* (denotes a specialisation of a concept, where the reverse name is *has-specialisation*), *instance-of* (denotes a special *is-a* relation in which an instance is an element of the extension of a concept, the reverse name being has-instances), *has-parts* (denotes a decomposition, subparts may be shared among concepts and the reverse name is *part-of*) and *has-decomposition* (denotes a decomposition where the sub-parts exist only if the surrounding part exists, the reverse name being *decomposition-of*). Other relationships may be defined by the users.

4. Specification and Conceptualization of the Ontology

In this section we describe the different stages undertaken in order to develop a semi-formal ontology which represents the main concepts that the standards and literature showed in Table 2 indicate that should be taken into account to model the SMP. The planning and implementation stages are omitted, the former being where the goals of the ontology and preliminary ideas for designed were proposed. The aspects related to the latter will be described in future papers, since this paper is focused on the design of the ontology.

4.1. *Specification*

In this initial phase the domain modelled, the purpose of the ontology, the scope, and administrative information, such as the authors and knowledge sources are indicated. Due to the SMP's complexity, several ontologies and subontologies have been developed. Thus, although the major concepts are normally listed in the scope row, we preferred to list the different ontologies and subontologies designed (see Table 1).

Table 1. Specification table.

Concept	Value
Domain	Management of Software Maintenance Projects
Author	Alarcos Research Group (UCLM)
Purpose	Ontology for enabling information interchange among engineers, managers and users of maintenance projects.
Level of formality	Semi-formal (REFSENO and UML)

Scope	List of concepts:
	This is classified (for reasons of clarity) into partial ontologies and subontologies:
	• Maintenance Ontology
	  ○ Products Subontology
	  ○ Activities Subontology
	  ○ Process Organization Subontology
	  ■ Procedures
	  ■ Requests Management
	  ■ Problems
	  ○ Agents Subontology
	• Workflow Ontology
	• Measurement ontology

Source of knowledge	See Table 2

The sources of knowledge on which we have based ourselves to develop this ontology have been the experience obtained from six R&D projects (see section 5) developed in collaboration with several national and international companies throughout six years, plus the sources enumerated in Table 2.

4.2. *Conceptualisation*

After the requirements specification has been described, the ontologies themselves must be developed. In order to do this we have followed the steps recommended by [51]. The following sections describe the three ontologies (Maintenance, Workflow and Measurement) and their subontologies designed to represent the SMP.

4.2.1. *Maintenance ontology*

The main sources of knowledge used to develop this ontology were D1, D2, D3, D4 and D5 (see Table 2), stressing the informal ontology proposed by [29], which

Table 2. Sources of knowledge utilised to construct the ontology.

D1		Informal ontology for SMP proposed by Kitchenham *et al.* [29].
D2		Conceptual model for the corrective maintenance by Kajko-Mattsson [26, 27].
D3		Ontology for the software development process proposed by Falbo *et al.* [10].
D4		Conceptual model for software process and software measurement proposed by [5].
D5		Documents which define the MANTIS processes system:
	D5a	– Model of ISO 12207 life cycle
	D5b	– Process reference model ISO15504-2 [23].
	D5c	– ISO 14764 about SMP model [24].
	D5d	– Model of activities and tasks of the MANTEMA methodology [43].
D6		Several workflow representation models:
	D6a	– Reference model of *Workflow Management Coalition* [55].
	D6b	– Conceptual model of Sadiq and Orlowska [49].
D7		Models for the measurement process:
	D7a	ISO 15939 for the measurement process [25].
	D7b	Conceptual model to represent software data collections, proposed by [28].

helped us to divide this ontology into four subontologies, one per domain that influences the SMP: **products** to be maintained, **activities** to be performed in order to maintain the products, **people** who are involved during the SMP and **process organisation,** which indicate how to carry out the activities. In these four subontologies four types of elements are represented: artefacts, activities, resources and agents, which are indispensable in the management of projects [44].

Products subontology

This subontology defines the software products that are maintained, their internal structure, composition and the existing versions of each product. Figure 2 shows the ontologic diagram, where the product is stressed since it is the most important one.

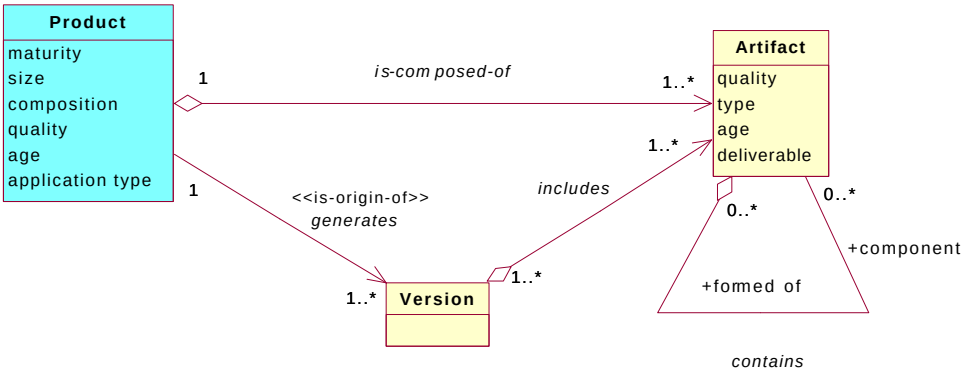

Fig. 2. Products subontology diagram.

Table 3. Products subontology: concept glossary.

Concept	Super-concept	Description	Purpose
Artifact	Element	This is a software product, part of which is created or modified by the activities. It can be a document (text or graphic), a COTS, or a code module. Examples: requirement specification documents, quality plan, class module, routine, test inform, user manual. Synonymous: software element, work product, product item.	To define the internal structure and software composition.
Product	Concept	Software application, which is being maintained. It is a conglomerate of different artifacts. Synonymous: Software.	Maintenance.
Version	Concept	This is a change in the base line of a product. It could be an *upgrade*, *release* or *actualisation*.	To implant the configuration management process.

NOTE: The super-concept "Concept" is the root.

As Fig. 2 shows, one software product can have different versions, which are formed from a set of artifacts. For instance, for a product called "Accounts", different versions of this product may exist, and each version is made up of several artifacts. The concept version has its own attributes, such as: number, date, etc. To simplify, they are not represented in the diagram.

The previous diagram only shows a summarized view of the referred ontology. The concept glossary contains the concepts previously represented in the diagram. Each row of the table corresponds to one concept.

In Tables 4 and 5 the terminal attributes of the artifact and product are represented respectively. As was explained in Sec. 3.1 attributes are typed and users can define new types. Tables 4 and 5 shows some types defined by us, such as MeasureQ, which defines a range of measures, or SwProductTaxo which makes up a taxonomy of software products.

The items value inference, inferred attributes and default value are omitted since on most occasions these columns do not have values. For this reason we consider that the table shown is the clearest, since it has fewer columns, and it has no loss of meaning.

Tables 4 and 5 describe the attributes of the concepts "Artifact" and "Product" respectively. The attributes of the concept version have been omitted in order to simplify the paper. Attributes preceded by (I/F) mean attributes of the interface layer. They are characterised by indicating how a particular instance can be integrated into the system. In the case of the "Artifact", the "source" attribute explains from which product the artifact is a subpart. The column labelled "Type" in this case, specifies the relationship between both concepts. Second and third interface

Table 4. Products subontology: attributes table for the concept "Artifact".

Concept: Artifact					
Name	Description	Cardi-nalinality	Type	Mandatory	Standard weight
Quality	Qualitative measure of the quality, especially of the artifact documentation.	1	MeasureQ	Yes	1
Type	Artifact Nature. Examples: documents, module, component, file.	1..*	SwProductTaxo	Yes	1
Age	Number of years from when the first version was obtained.	1	Integer	Yes	1
Deliverable	This indicates whether the artifact must be delivered to the client or not.	1	Boolean	Yes	1
(I/F) source	This indicates the product of which it is a part.	1	Decomposition-of [product]	Yes	1
(I/F) artifacSon	This indicates into which artifacts it is divided.	0..*	Has-parts [artifact]	No	1
(I/F)artifacFather	This indicates of which artifact it is a part.	0..1	Part-of [artifact]	No	1

attributes show whether the artifact has sub-parts and whether it is the sub-part of another artifact.

The rest of the attributes belong to the artifact layer that characterises the instance itself. Examples of these attributes are quality, type, age or deliverability, in the case belonging to the concept called "Artifact".

For the case of the "Product", there is only one interface layer attribute which indicates in which artifacts the product may be decomposed. Obviously, if the artifact has a relationship *Decomposition of* with the concept "product", this concept must have the opposite relationship *Has-decomposition*.

Taking advantage of the information shown in Table 5 we are going to illustrate how the similarity function for the concept "Product" would be calculated. First of all, the similarity functions for each layer, artefact and I/F (the context layer is omitted because in this case there are not attributes of this layer) should be calculated. To calculate each local similarity function it is necessary to know the similarity function associated with each type of the attributes of each layer. Thus, in the case of the artifact layer of the concept product, it is necessary to know the similarity function of the types "TypeMaturity", "MeasureSize", "TypeComposition", "TypeApplication", "MeasureQ" and "Integer". REFSENO provides several

Table 5. Products subontology: attributes table for the concept "Product".

Concept: Product					
Name	Description	Cardi-nalinality	Type	Mandatory	Standard weight
Maturity	Phase of the product in the life-cycle.	1	TypeMaturity	Yes	1
Size	Qualitative measure of the size.	1	MeasureSize	Yes	1
Composition	Abstract level of the artifacts that form it.	1	TypeComposition	Yes	1
Application type	Type of application. For instance: management, scientist, etc.	1	TypeAplicaction	Yes	1
Quality	Qualitative measurement of the quality, basically of the documentation.	1	MeasureQ	Yes	1
Age	Number of years from when the first version was obtained.	1	Integer	Yes	1
(I/F)Component	Artifact into which the product is divided.	0..*	Has-decomposition [artifact]	No	1

predefined types and their similarity functions. For instance, the "Boolean" type which has two values, true and false, has the following similarity function for comparing two instances i and q:

$$\begin{aligned} \mathrm{Sim}(i,q) = 1 \quad & \text{when } i = q \\ 0 \quad & \text{otherwise.} \end{aligned}$$

In the case of using own types, such as in "TypeMaturity" or "MeasureQ", their similarity types should also be described. Then, the local similarity functions are calculated by computing the sum of the similarity function of each type of attribute belonging to this layer. Finally, each local similarity function is normalized resulting in a value in the range [0,1].

In order to know the global similarity function, different values should be assigned to W_{artif}, and $W_{\text{I/F}}$, depending on what the needs of user are. For instance, if a user wants to compare the similarity between two products according to their own features, the value of W_{artif} should be maximised. For more information about the similarity functions see [51].

These functions are very useful in the design of CBR techniques, which KM-MANTIS uses. Concretely, KM-MANTIS uses similarity functions to compare software products and maintenance requests in order to detect similarities between new demands and previous ones, with the goal of reusing information and proven solutions, thus decreasing costs and effort. CBR techniques have also been successfully used in software reuse in [6, 14] and in other topics related to reuse in [34].

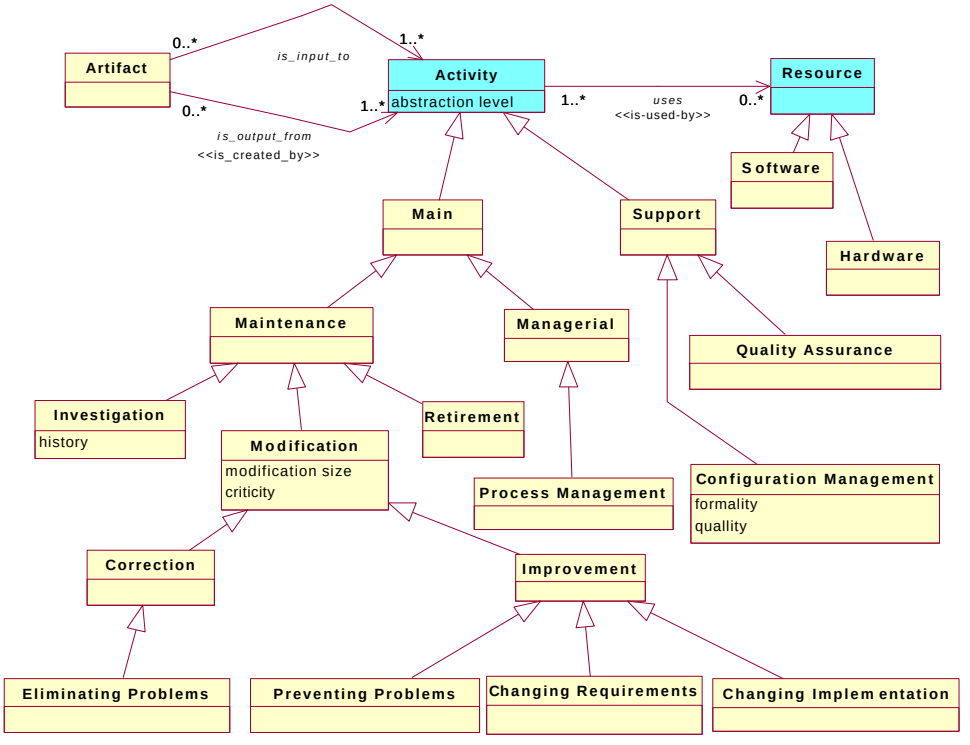

Fig. 3. Activity subontology diagram.

Activity subontology

This subontology includes two of the four critical elements for managing a maintenance project: activities and resources. The activity subontology defines: a taxonomy for the types of activities, a taxonomy for the types of resources and also defines the relationship between artifacts, activities and resources.

Figure 3 shows that the activities are classified into "Main" and "Support" activities, and they are also divided into different types. In the diagram the most important ones have been highlighted. For example "Managerial" is a "Main" activity (sub)type, which may be a "Process Management" activity type.

The maintenance activities are classified, according to the ISO 14764 [24], into "Investigation", "Modification", and "Retirement" activities. The modification activities consist basically of two types: "Correction" and "Improvement". The first one belongs to corrective maintenance, where mistakes are eliminated. The second one is in charge of preventing problems (preventive maintenance), implementing changes in the requirements (perfective maintenance) or changing aspects of implementation (adaptive maintenance) [48]. For instance, to change the operative system used by the application, but without changing the application requirements.

The dependencies of execution between the activities are not included in this ontology, since we considered that this aspect was more related to the workflows, therefore they are defined in the workflow ontology.

The concept glossary for the subontology activity and the attributes table for the concept "Activity", are shown in the following tables:

Table 6. Activities subontology: concepts glossary.

Concept	Super-concept	Description	Propose
Activity	Element	Action which is carried out in order to achieve the project's goals. Synonyms: Task, step	Describing the work to be performed
Artifact	–	See products subontology	–
Quality Assurance	Support	Support activities performed to make sure that the processes and products follow the requirements and established plans.	Guaranteeing the quality of the product
Changing Implementation	Improvement	Improvement activity carried out to adapt a software product to changes in its implementation environment without affecting the requirements.	Giving maintenance service
Changing Requirements	Improvement	Improvement activity carried out to adapt software working to changes in the requirements, or to the inclusion of new requirements. Perfective maintenance.	Giving maintenance service
Correction	Modification	Modification activity which consists of eliminating the defects in a software product so that it will work as requirements indicate.	Giving maintenance service
Eliminating Problems	Correction	Correction activity carried out to eliminate detected problems. Synonym: Corrective maintenance.	Giving maintenance service
Maintenance	Main	Activity to manage the SMP. This activity is included in the organisational subsystem, defined in the MANTIS process system.	Managing the SMP
Configuration Management	Support	Support activity whose objective is to establish and maintain the integrity of said artifacts, and make them available via different versions to the projects agents.	Guaranteeing the quality of the versions
Process Management	Managerial	Activity to organise and monitor the initialisation and carrying out of SMP.	Managing the SMP
Hardware	Resources	Resource formed from a computer science system, a computer and a peripheral artifact.	Utilising the computer science artifact

Table 6. (*Continued*)

Concept	Super-concept	Description	Propose
Investigation	Maintenance	Activity which checks the different ways of carrying out a maintenance requirement, and its impact on the software product.	Giving maintenance service
Managerial	Main	Activity included in the SMP according to the ISO/IEC [24].	Performing the SMP
Improvement	Modification	Activity which implements changes in a software product. The changes modify the product's behaviour or improve the quality features.	Giving maintenance service
Modification	Maintenance	Activity which creates or modifies one or several artifacts, changing the behaviour or implementation of the product.	Giving maintenance service
Preventing Problems	Improvement	Activity which eliminates problems even though they have not been considered defects. Synonym: Preventive maintenance.	Giving maintenance service
Main	Activity	Activity belonging to the main subsystem, or to the organisational subsytem defined in the process system of MANTIS.	Performing and managing the SMP
Resource	Element	Something that is necessary for performing an activity, but is not part of the software product.	Managing resources
Retirement	Maintenance	Activity which is performed to terminate the life of a software product.	Terminating a maintenance service
Software	Resource	Software tool used by the total or partial automatisation of some activities.	Using the software tool
Support	Activity	Activity whose goal is to facilitate the carrying out of the main activities.	Giving support to the main activities

Table 7 represents the fact that one activity can be composed of several subactivities, or on the contrary, that one activity is part of another more complex activity. They are attributes belonging to the interface layer.

Process organization subontology

This subontology includes the concepts which define how to carry out the different activities. It also indicates how the maintenance process itself is organised.

One of the main differences between the phase of software development and the phase of maintenance is that the former is directed by requirements and the latter is directed by events. This means that the inputs triggered by the maintenance activities are non-plannable random events. These are the maintenance requests

Table 7. Activities subontology: attributes table for the "Activity" concept.

Concept: Activity					
Name	Description	Cardinality	Type	Mandatory	Standard weight
Abstraction level	This indicates the degree of abstraction of an activity within the processes system taxonomy.	1	TaxonoPro	Yes	1
(I/F)Subactivity	Activities that must be performed to carry out the activity.	0..*	Has-parts [activity]	No	0
(I/F)ActivityFather	Activity which depends on the activity.	0..*	Part-of [activity]	No	0

(MR). A critical aspect for a maintenance organization is to appropriately manage the queue of requests that it receives.

In this subontology it is possible to distinguish three parts focused on the following issues:

— Identification of problems and their types
— Procedures to carry out activities
— Maintenance and management activities which make it possible to manage MR

Identification of problems and their types

Investigation activities produce reports related to the MR. These reports, called investigation reports contain information about the different problems found in the software. An investigation report is an artifact of document type, which identifies the reasons why a problem arises (the problem has previously been communicated by an MR), and the effects that it produces in the software. An investigation report can be related to a problem which has already been analysed in previous reports, and in this case the investigation report complements the previous one.

The reasons for a problem arising could be within the software itself, or be produced by other causes. The most common reasons related to the software found are failures, or incorrect operations that have taken place during the execution [26].

Procedures to carry out the activities

One activity can be carried out by using one or more procedures. That does not mean that all the procedures are always used, because it is possible that some procedures have the same means of functioning. In this case one must be chosen.

For each procedure, its type (method, technique, script) and its constraints must be defined according to the paradigm and technology used [10]. The software tools useful to automate each procedure and the artifacts modified or created by the procedure should also be reflected in the ontology.

Maintenance requests management

Figure 4 shows part of the subontology which represents the main activities and artifacts in the software maintenance process. The process management consists of a collection of activities which are in charge of managing the MR, and another set of activities dedicated to the control of changes. These activities use basically two types of artefacts: MR, which trigger the activity, and the investigation reports. There are two types of MR depending on their contents. They can indicate the

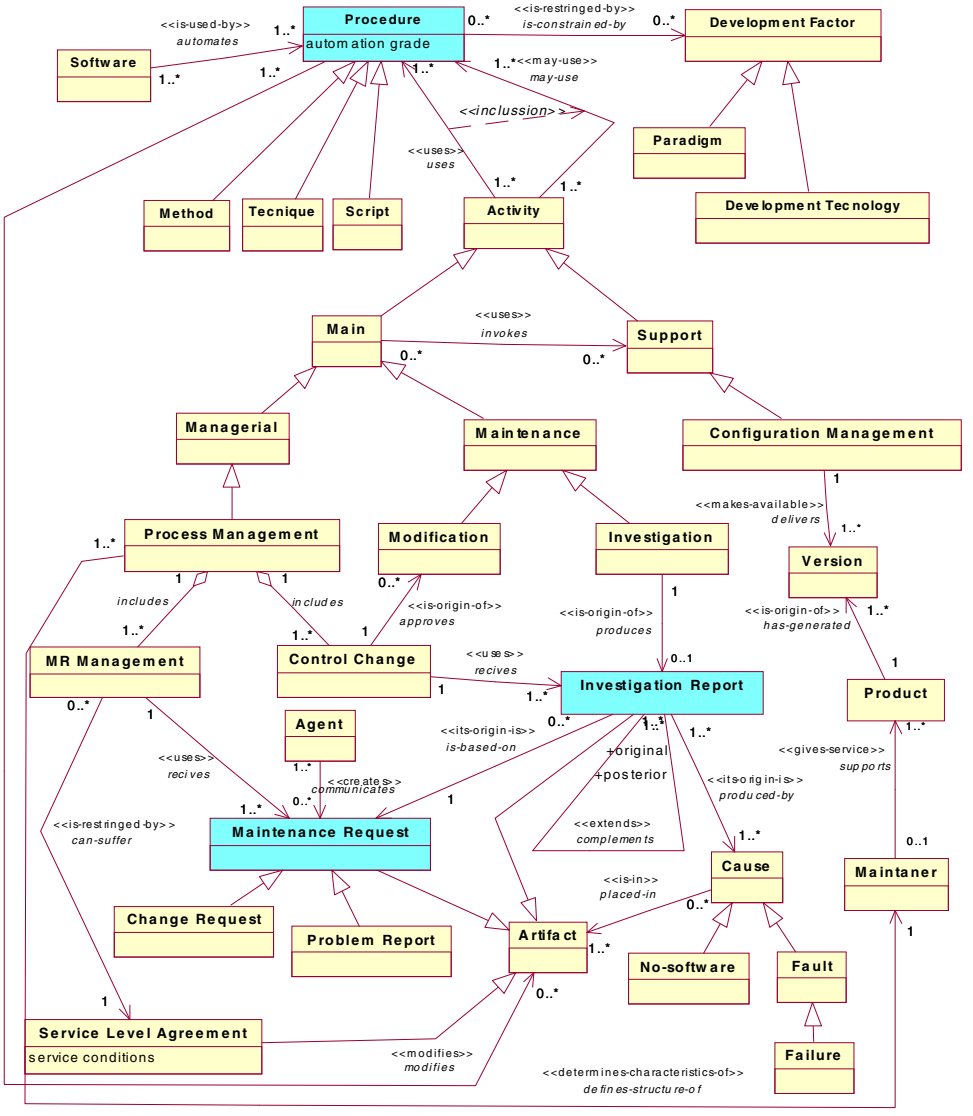

Fig. 4. Process organization subontology.

description of a problem detected in the software (problem report), or a request for a change (change request).

When the MR management receives an MR, analyses which the organisation agreed with the client, according to the "Service Level Agreement". In the case that the MR is within the services established, an "Investigation Report" is generated from an "Investigation" activity. This report is received by the "Control Change" activity, which is in charge of deciding which modification activities are accepted (in the case of there being any).

One aspect to take into account is the support activities. These are invoked by the process management or by the maintenance activities. Within the support activities is the configuration management, which has strong repercussions on the quality and performance of the maintenance service because it is in charge of delivering the new versions of the product to users. Figure 4 shows the process organisation subontology, where the three aspects explained (problems, procedures, and MR management), are integrated.

This, and the following subontologies, have concept tables similar to those presented for the previous subontology. However, with the end result of not making this paper too long, in this subontology and in the following ones, only the UML diagrams are shown.

Agents subontology

Different agents are involved in the SMP. We have classified them in the following way:

(a) Automatic agents (software tools) and human agents.
(b) Human agents can be people (individual) agents or organisations. The latter consists of people. Each person plays a determinate role in each organisation [5].
(c) Each organisation has an organisational model, which is represented by a hierarchy of aggregations formed of the subordinate organisations.
(d) In accordance with the MANTEMA methodology, there are three organisations: maintainer, client and user [42]. Sometimes the client and the user are the same, but the most frequent aspect is that they are not.

In Fig. 5 we can see that the model of agents is based on the concept of role. This allows generalisation and flexibility to represent any possible situation which takes place in a real project [43]. Therefore, during the process definition, the roles and their responsibilities are established. On the other hand, it is during the project planning stage, where the agents are involved, and their roles are indicated.

4.2.2. *Workflow ontology*

Recently, some authors [38] have suggested using workflow for dealing with software processes, thus taking advantage of the existing similarity between the two

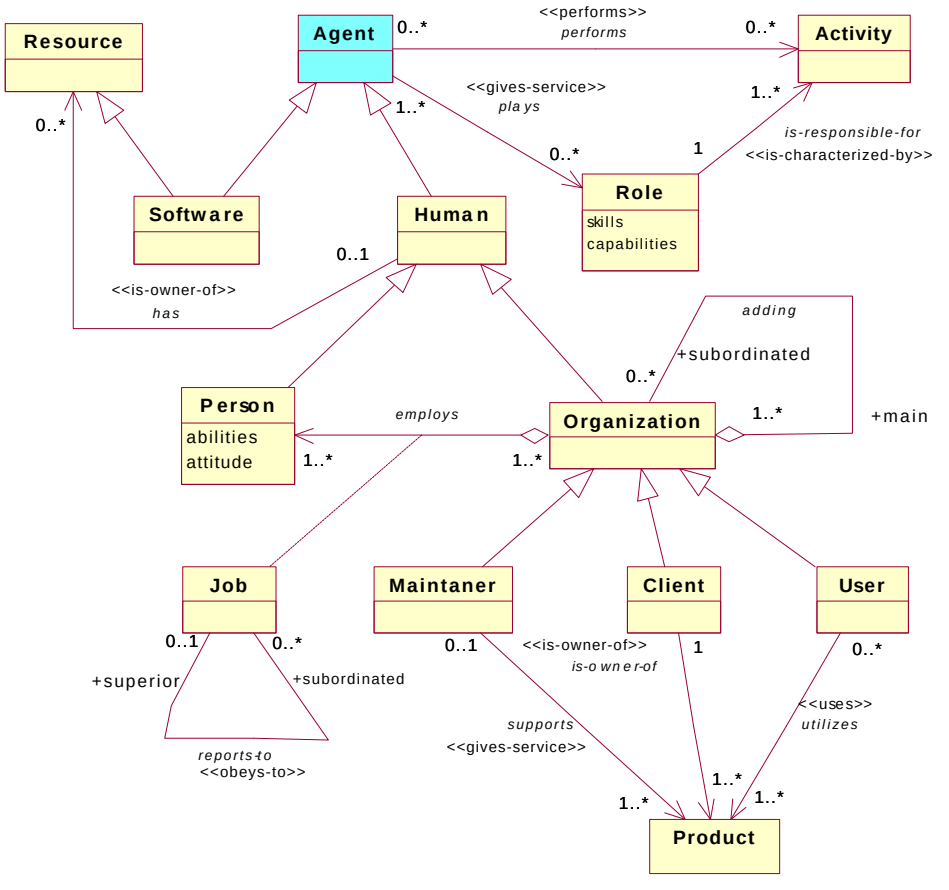

Fig. 5. Agents subontology diagram.

technologies. The value of *Workflow Management Systems* (WfMS) in the automation of business processes has been clearly demonstrated, and given that SMP can be considered as part of a wider business process, it is reasonable to consider that workflow technology will be able to contribute a broader perspective to SMP.

These reasons have led us to integrate workflow concepts in the ontology. We have incorporated aspects of the "Workflow Reference Model", of the Workflow Management Coalition [56], and aspects of other proposals [31].

The Workflow ontology incorporates aspects corresponding to the following three issues [47]:

(a) Decomposition of a complex activity into more simple activities.
(b) Constraints between activities, in other words, the order in which activities are performed.
(c) Controlling the state of the activities and projects during the processes enactment.

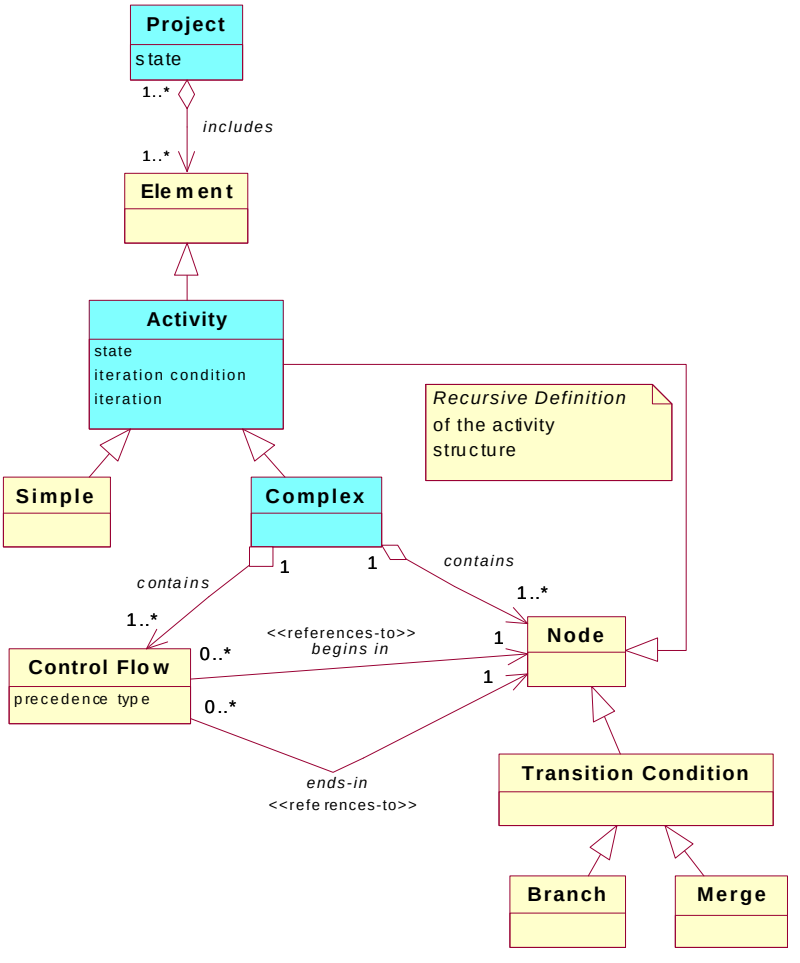

Fig. 6. Workflows ontology diagram.

The first aspect has been modelled on the activity ontology. However, in this ontology we have added the concept of "Workflow". Thus, according to the workflow model presented by [49] two types of activities exist: "Simple" (those that do not have an internal structure related to the project management), and "Complex" (they imply the execution of a workflow). A complex activity contains a set of interrelated "Nodes" and "Control flows". Nodes could be activities (sub-activities belonging to a complex activity) or "Transition conditions". Therefore, a recursive definition of the structure of an activity is used. For example, a main activity has an internal structure modelled by a workflow, that has sub-activities, which might also be complex and be modelled by another workflow, and so on until we obtain simple sub-activities. Transition conditions could be "Branch" or "Merge".

On the other hand, a control flow represents a precedence relationship between

two nodes, identified by the associations called "begin-in" and "finish-in". A control flow may be of different types depending on the kind or precedence that it represents: "finishing to start", "start to finish", "start to start", etc.[a]

The iterative structures (loops that repeat the execution of an activity more than once) are represented by the activities attributes, called "iteration" and "condition iteration". Activities with iteration mean that there is the possibility that several different enactments of the same activity exist during the enactment of an instance of a project.

The state of execution of a project and its corresponding activities are represented in the ontology by an attribute called "state". The different possible states are: Not started, in execution, suspended, finished, and aborted [55]. Figure 6 shows all the considerations explained.

4.2.3. *Measurement ontology*

A fundamental aspect to manage and control in any project, is the availability of a set of metrics which will allow the measurement of the product that is being produced or maintained, and also how the project is being executed. Both aspects are fundamental for the quality assurance and the assessment or improvement of the process. For these reasons, and for software engineering to be considered as such, it is essential to be able to measure what is being done. For this purpose, the Measurement ontology includes the concepts of measurement, metrics, values and attributes associated with the activities, artifacts and resources. This is bearing in mind that the measurements can refer to both product and process aspects. This measurement ontology is based on the process measurement standard ISO 15939 [25]. This standard claims that it is necessary to define metrics (named measurement, noun) for the process. These metrics will help to satisfy the organization information necessities.

According to the standard, all that can be measured are "entities". An entity is an object such as a process, a product, a project or a resource, which may be characterised through its attributes. All attributes are associated to a metric which is linked to a measurement unity (for instance, number of code lines), and a measurement unity in turn belongs to a determinate scale. According to the standard, four scales may be differentiated: nominal, ordinal, interval and ratio, although it is possible to establish other classifications such as Kitchenham's *et al.* [28].

The measurement ontology (see Fig. 7) shows three types of metrics:

- **Base metric**. This is defined from an attribute and the necessary method to quantify it.

[a]These are the types of precedence relationships defined by PMI [38] and used in the majority of the systems for project management. However, it is possible to define other types by using other different constraints.

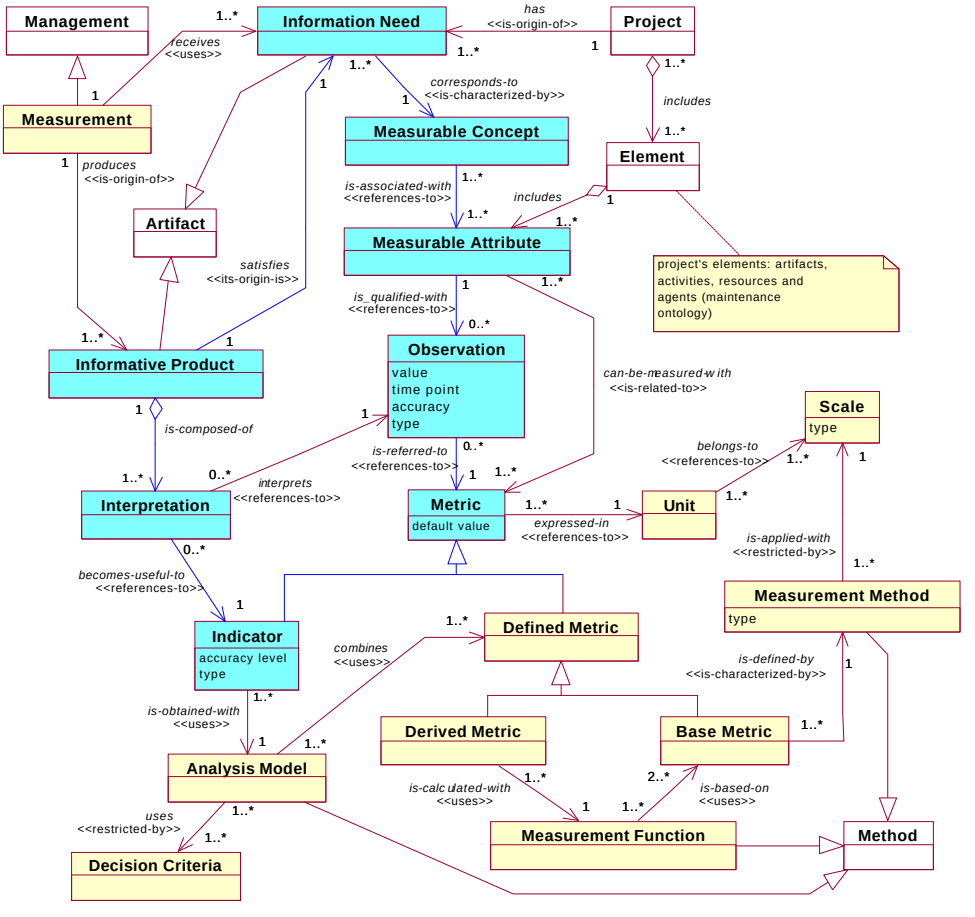

Fig. 7. Measurement ontology diagram.

- **Derived metric.** This is defined from two or more base measure values.
- **Indicator.** The measure that gives an estimation of the attributes derives from a model. Indicators are the basis for analyses and decision-making.

The measurement process is directed by information necessities [15]. For each information necessity the process creates one informative product that fulfills that necessity. Informative products are the base for making decisions in organisations. For more information about the measurement ontology see [16].

5. Using the Ontology

The nature of an ontology implies that it cannot be validated with empiric methods. For this reason, for the development of the ontology presented in this paper, we have used Action-Research [9], a qualitative research method that has been very useful to us thanks to its main characteristics. The method focuses on a problem,

requires an iterative process, and mandatory collaboration between researchers and stakeholders. Thus, the presented ontology is largely the result of the acquired experience of the collaboration with several organizations who showed interest in the software maintenance service field. The strongest collaboration has been with Atos ODS Origin, a European leading company in the outsourcing of software services with a billing of over 6000 million Euros.

This collaboration has been developed within the framework of several R&D projects whose main objectives have been to:

— define a specific methodology for maintenance (MANTEMA, 1997–2000);
— establish new methods to improve the SMP (MPM, 1998–2000);
— define metrics of maintainability (MANTICA, 1999–2001);
— design and develop an "extended software engineering environment" to manage maintenance projects (MANTIS, 2000–2001);
— improve the SMP by mean of software measurement and knowledge management (TAMANSI, 2002–2004);
— develop new test techniques for the agile maintenance of software (MAS, 2003–2005).

Thus, from an initial idea focused only on the aspects of carrying out maintenance tasks (MANTEMA project), we enlarged and revised the ontology, continually collaborating with stakeholders, until the current general proposal was defined.

The global version of the ontology that we have summarized in this paper has been and is being used in a successful way in several research works that we shall mention briefly:

(a) During the development of the MANTIS project the necessity of having a software maintenance ontology was detected since important problems of conceptual and terminological character arose. These problems hindered the correct collaboration with stakeholders and the interaction with other researchers. The main lesson learnt was: if you want to manage software maintenance projects from a wide business process point of view (integrating software engineering aspects with other managerial ones), you need an ontology to facilitate sharing knowledge among all those who participate in the SMP [45]. More concretely, the existence of this ontology was of great help in implementing the MANTIS environment. It served as a "filter of knowledge" when creating the models and metamodels of processes and application domains [19]. It has also been very useful when establishing the functional requirements of the MANTIS global repository [46].
(b) At present, the ontology is being used in the TAMANSI project for the construction of KM-MANTIS, a knowledge management system to improve maintenance project management [54]. With KM-MANTIS, we intend to transform the MANTIS environment into an integrated framework and into a collection of software tools useful to attain levels 4 (managed) or 5 (optimized) of CMM.

In this sense, the availability of the ontology has made it simpler to define the functional requirements of KM-MANTIS. It is also of great help in the design and implementation of this new software component.

(c) The above measurement ontology is also being used to develop a software processes measurement framework and of a tool called Gen-METRIC [15]. Taking advantage of the generality of the ontology, we seek to define, and to be able to measure, by using the tool, abstract metrics or meta-metrics that define families of metrics of the same nature. For example, the "number of tables in a relational schema" or the "number of modules in one product software". In a similar way as in the MANTIS project, the ontology has been the key during the creation of the metamodels and models of processes, software products and metrics.

6. Conclusions

This paper presents a summary of a semi-formal ontology for managing software maintenance projects. In this ontology, the main aspects, according to the software process standards, have been considered: products, activities, processes, agents, measures, and some dynamic aspects (workflows). This ontology is different from previous ones which have focused on static features and have been highly centred on a particular problem.

Another important issue is the practical character of this ontology, since it has already been used in several projects. This fact helped us to detect some of its limitations and to improve them. For instance, Workflows were added when we discovered that it was very important to complete the ontology. Moreover, the last version of the ontology is currently being used in the development of a new KM module, in order to manage and reuse information and knowledge generated during maintenance projects management (see [54]). Having a clear and precise ontology was critical to the construction of an intelligent system, since a common conceptualisation facilitates the sharing and reuse of information.

To develop the ontology, a suitable methodology has been used, first defining the concepts involved, second the attributes of each concept and third the relationship between the different concepts. An additional step to control overlaps was carried out. The ontology has been represented by using REFSENO, formalism for software engineering ontologies.

Acknowledgements

The authors would like to thank the anonym reviewers for their useful recommendations to improve this paper.

This work is partially supported by the TAMANSI project (grant number PBC-02-001, Consejería de Ciencia y Tecnología, Junta de Comunidades de Castilla-La Mancha) and the MAS project (grant number TIC2003-02737-C02-02, Ministerio de Ciencia y Tecnología, SPAIN).

References

1. K.-D. Althoff, A. Birk, S. Hartkopf, and W. Müller, Managing software engineering experience for comprehensive reuse, in *Proc. 11th Int. Conf. on Software Engineering*, Kaiserslautern, Germany, 1999.
2. K.-D. Althoff, A. Birk, and C. Tautz, The experience factory approach: Realizing learning from experience in software development organizations, in *Proc. 10th German Workshop on Machine Learning* (*FGML'97*), University of Karlsruhe, 1997, pp. 6–8.
3. V. R. Basili, G. Caldiera, and H. D. Rombach, The experience factory, in *Encyclopedia of Software Engineering*, ed. J. J. Marciniak, (John Wiley & Sons, 1994), pp. 469–476.
4. V. R. Basili and H. D. Rombach, The TAME project: Towards improvement-oriented software environments, *IEEE Trans. on Software Engineering* **SE-14**(6) (1988) 758–773.
5. U. Becker-Kornstaedt and R. Webby, A Comprehensive Schema Integrating Software Process Modelling and Software Measurement, Fraunhofer IESE-Report No. 047.99 (Ed.: Fraunhofer IESE, 1999), http://www.iese.fhg.de/Publications/Iese_reports/.
6. R. Bergmann and U. Eisenecker, Case-based reasoning for supporting reuse of object-oriented software: A case study, in *Proc. Expert Systems* **95** (1996) 152–169.
7. D. N. Card and R. L. Glass, *Measuring Software Design Quality* (Prentice Hall, Englewood Cliffs, NJ, 1990).
8. D. Deridder, A concept-oriented approach to support software maintenance and reuse activities, in *Proc. Workshop on Knowledge-Based Object-Oriented Software Engineering at 16th European Conference on Object-Oriented Programming* (*ECOOP 2002*), Málaga, Spain, 2002.
9. C. Estay and J. Pastor, Improving action research in information systems with project management, in *Proc. Americas Conference on Information Systems*, Long Beach, CA, USA, 2000, pp. 1558–1561.
10. R. A. Falbo, C. S. Menezes, and A. R. Rocha, Using ontologies to improve knowledge integration in software engineering environments, in *Proc. 4th Conference on Information Systems Analysis and Synthesis*, Orlando, Florida, USA, 1998.
11. R. A. Falbo, C. S. Menezes, and A. R. Rocha, Using knowledge serves to promote knowledge integration in software engineering environments, in *Proc. 11th Int. Conf. on Software Engineering and Knowledge Engineering* (*SEKE'99*), Kaiserslautern, Germany, 1999.
12. A. Farquhar, R. Fikes, and J. Rice, The Ontolingua server: A tool for collaborative ontology construction, *Int. J. Human-Computer Studies* **46** (1997) 707–728.
13. C. Fernández, A. Gómez-Pérez, and N. Juristo, METHONTOLOGY: From ontological art towards ontological engineering, in *Proc. AAAI Spring Symposium*, University of Stanford, Palo Alto, California, USA, 1997, pp. 33–40.
14. M. Fernández-Chamizo, P. A. González-Cálero, M. Gómez-Albarrán, and L. Hernández-Yánez, *Supporting Object Reuse through Case-based Reasoning* (Springer-Verlag, 1996), pp. 135–149.
15. F. García, F. Ruiz, J. A. Cruz, and M. Piattini, Integrated measurement for the evaluation and improvement of software processes, in *Proc. 9th Int. Workshop Software Process Technology,* (*EWSPT'2003*), Helsinki, Finland, 2003, pp. 94–111.
16. M. Genero, F. Ruiz, M. Piattini, and C. Calero, Towards and ontology for software measurement, in *Proc. Int. Conf. on Software Engineering and Knowledge Engineering* (*SEKE'2003*), San Francisco, USA, 2003.
17. A. Gómez-Pérez, *Knowledge Sharing and Reuse* (CRC Press, 1998).
18. T. Gruber, Towards principles for the design of ontologies used for knowledge sharing, *Int. J. Human-Computer Studies* **43**(5/6) (1995) 907–928.

19. M. Gruninger and J. Lee, Ontology applications and design, *Commun. ACM* **42**(2) (2002) 39–41.
20. N. Guarino, Formal ontology and information systems, in *Proc. 1st Int. Conf. on Formal Ontologies in Information Systems, FOIS'98*, Trento, Italy, 1998.
21. T. Hikita and M. J. Matsumoto, Business process modelling based on the ontology and first-order logic, in *Proc. 3rd Int. Conf. on Enterprise Information Systems* (*ICEIS'2001*), Setubal, Portugal, 2001, pp. 717–723.
22. IEEE (1995), STD 1074-1995: IEEE Standard for Developing Software Life Cycle Processes.
23. ISO/IEC, 15504-2: *Information Technology-Software Process Assessment-Part 2: A Reference Model for Processes and Process Capability*, 1998.
24. ISO/IEC (1998), FDIS 14764: Software Engineering-Software Maintenance (draft), Dec. 1998.
25. ISO/IEC (2002), FDIS 15939: Software Engineering-Software Measurement Process (draft), Jan. 2002.
26. M. Kajko-Mattsson, Common concept apparatus within corrective software maintenance, in *Proc. IEEE Int. Conf. on Software Maintenance* (*ICSM'99*), Oxford, UK, 1999, pp. 287–296.
27. M. Kajko-Mattsson, Towards a business maintenance model, in *Proc. IEEE Int. Conf. on Software Maintenance* (*ICSM*), Florence, Italy, 2001, pp. 500–509.
28. B. A. Kitchenham, R. T. Hughes, and S. G. Linkman, Modelling software measurement data, *IEEE Trans. on Software Engineering* **27**(9) (2001) 788–804.
29. B. A. Kitchenham, G. H. Travassos, A. Mayrhauser, F. Niessink, N. F. Schneidewind, J. Singer, S. Takada, R. Vehvilainen, and H. Yang, Towards an ontology of software maintenance, *J. Software Maintenance: Research and Practice* **11** (1999) 365–389.
30. M. Klein, Combining and relating ontologies: An analysis of problems and solutions, in *Proc. Workshop on Ontologies and Information Sharing* (*IJCAI'2001*), Seattle, USA, 2001.
31. C. Liu, X. Lin, X. Zhou, and M. Orlowska, Building a repository for workflow systems, in *Proc. 31st Int. Conf. on Technology of Object-Oriented Language and Systems*, 1999, pp. 348–357.
32. L. A. Loof de, *Information Systems Outsourcing Decision Making: A Managerial Approach* (IDEA Group Publishing, Hershey, PA, 1997).
33. A. Maedche, B. Motik, L. Stojanovic, R. Studer, and R. Volz, Ontologies for enterprise knowledge management, *IEEE Intelligent Systems*, 2003, pp. 26–33.
34. L. McGinty and B. Smyth, Collaborative case-based reasoning: Applications in personalised route planning, in *Proc. 4th Int. Conf. on Case-Based Reasoning*, Berlin, 2001.
35. D. L. McGuinness, R. Fikes, J. Rice, and S. Wilder, *An Environment for Merging and Testing Large Ontologies*, 2000, pp. 483–493.
36. J. Mylopoulos, Information modelling in the time of the revolution, *Information Systems* **23**(3-4) (1998) 127–155.
37. J. Mylopoulos, *Ontologies*, http://www.cs.toronto.edu/∼jm/2510S/Notes02/Ontologies.pdf. Visited on 4th November, 2002., 2001.
38. C. Ocampo and P. Botella, Some reflections on applying workflow technology to software processes, Universitat Politecnica de Catalunya, Computer Science Systems Department, Technical Report TR-LSI-98-5-R, Barcelona, Spain, 1998.
39. E. Ostergag, J. Hendler, R. Prieto-Díaz, and C. Braun, Computing similarity in a reuse library system: An AI-based approach, *ACM Trans. on Software Engineering and Methodology* **1**(3) (1992) 205–228.

40. T. M. Pigoski, *Practical Software Maintenance. Best Practices for Managing Your Investment* (John Wiley & Sons, 1997).
41. H. S. Pinto and J. P. Martins, Ontology integrations: How to perform the process, in *Proc. Workshop on Ontologies and Information Sharing* (*IJCAI'2001*), Seattle, USA, 2001, pp. 71–80.
42. M. Polo, M. Piattini, F. Ruiz, and C. Calero, Roles in the maintenance process, *Software Engineering Notes, Special Interest Group on Software Engineering, ACM* **24**(4) (1999) 84–86.
43. M. Polo, M. Piattini, F. Ruiz, and C. Calero, MANTEMA: A complete rigorous methodology for supporting maintenance based on the ISO/IEC 12207 standard, in *Proc. Third Euromicro Conference on Software Maintenance and Reengineering* (*CSMR'99*), Amsterdam, The Netherlands, 1999, pp. 178–181.
44. Project Management Institute, *PMBOK: A Guide to the Project Management Body of Knowledge*, 2000 edn., PMI Communications, USA, 2000.
45. F. Ruiz, F. García, M. Piattini, and M. Polo, *Environment for Managing Software Maintenance Projects*: Ideal Group Publishing, chapter X, 2002, pp. 255–290.
46. F. Ruiz, M. Piattini, F. García, and M. Polo, An XMI-based repository for software process meta-modelling, in *Proc. Product Focused Software Process Improvement* (*PROFES'2002*), Rovaniemi, Finland, 2002, pp. 546–558.
47. F. Ruiz, M. Piattini, and M. Polo, Using metamodels and workflows in a software maintenance environment, in *Proc. VII Argentine Congress on Computer Science* (*CACIC'01*), El Calafate, Argentina, 2001.
48. F. Ruiz, M. Piattini, M. Polo, and C. Calero, Types in the MANTEMA methodology, in *Proc. Int. Conf. on Enterprise Information Systems* (*ICEIS'99*), Setubal, Portugal, 1999, pp. 192–202.
49. W. Sadiq and M. E. Orlowska, On capturing process requirements of workflow based business information systems, in *Proc. 3rd Conference on Business Information Systems* (*BIS'99*), Poznan, Polonia, 1999, pp. 195–209.
50. S. Staab, H.-P. Schnurr, and Y. Sure, Knowledge processes and ontologies, *IEEE Intelligent Systems* **16**(1) (2001) 26–34.
51. C. Tautz and C. G. Von Wangenheim, *REFSENO: A Representation Formalism for Software Engineering Ontologies*, Fraunhofer IESE-Report No. 015.98/E, version 1.1, October 20, 1998.
52. M. Uschold and M. Gruninger, Ontologies: Principles, methods, and applications, *The Knowledge Engineering Review* **11**(2) (1996) 93–136.
53. G. Van Heijst, S. Falasconi, A. Abu-Hanna, T. Schreiber, and M. Stefanelli, A case study in ontology library construction, *Artificial Intelligence in Medicine* **7** (1995) 227–255.
54. A. Vizcaíno, F. Ruiz, J. Favela, and M. Piattini, A multi-agent architecture for knowledge management in software maintenance, in *Proc. Int. Workshop on Practical Applications of Agents and Multiagent Systems* (*IWPAAMS'02*), Salamanca, Spain, 2002, pp. 39–52.
55. WfMC (1999), TC-1016-P 1.1: Interface 1: Process Definition Interchange Process Model. Workflow Management Coalition, October 1999, http://www.wfmc.org/standards/docs.htm.,1999.
56. WfMC (1995), TC-1003 1.1: The Workflow Reference Model. Workflow Management Coalition, January 1995, http://www.wfmc.org/standars/docs.htm, 1995.