A Model-Driven Approach for the Improvement of Web Applications' Navigability

Abstract

Navigability in use, defined as the efficiency, effectiveness and satisfaction with which a user navigates through the system in order to fulfil her goals under specific conditions, has a definite impact on the overall success of Web applications. This quality attribute can be measured based on the navigational model provided by Web Engineering methodologies. Most of the measures currently defined for navigational models are tightly coupled with particular Web Engineering methodologies, however. Furthermore, modifications to the design of the navigational model, carried out with the aim of improving navigability, are performed manually. Both practices have seriously hampered the reusability and adoption of proposed navigability measures and improvement techniques. In this paper we present a Model-Driven Engineering approach to solving these problems. On the one hand, we propose a generic approach for the definition of navigability measurement models that can be integrated into any Web Engineering methodology. On the other hand, we present a model-driven improvement process for the navigational model design which incurs no increase in costs or in time-to-market of Web applications. This process is divided into two phases: evaluation (i.e. assessment of the model through objective measures) and evolution (i.e. transformation of the model when the measurement results do not fall inside the boundaries set by certain quality decision criteria that have been defined previously).

Introduction

The ever-increasing complexity of Web applications has caused the Web Engineering (WE) field, defined as the application of systematic, disciplined and quantifiable approaches to the cost-effective development and evolution of high-quality applications in the World Wide Web (Heuser, 2004), to evolve at an extremely fast pace. This discipline intertwines sound Software Engineering principles with a suitable set of models, particular to the idiosyncrasy of the Web. However, despite this definition, the inclusion in the different WE methodologies of mechanisms that will contribute to guaranteeing that the resulting applications fulfil a set of quality requirements continues to be a challenge for the discipline. In fact, there is no empirical evidence that there exists a significant relationship between the Web application quality in use and whether the designer follows a specific WE methodology for its development or not.

Even more perplexing for the WE field, a recent study shows that despite the fact that 84% of European enterprises use a hypermedia development process with clear tasks, phases and deliverables defined, only around 5% use or even know about the WE methodologies proposed in research (Lang and Fitzgerald, 2005).

If enterprises are to be encouraged to make the shift towards using a WE methodology, at least two WE promises must be fulfilled clearly and definitely:

- The use of WE methodologies should clearly decrease not only the development costs, along with time-to-market, but also the Web application maintenance and evolution costs, the latter making up over 50% of the total application costs in Software Engineering (Myers and Rosson, 1992).
- The use of WE methodologies should provide the designer with mechanisms which would guarantee that the Web application fulfills a set of both internal and external quality requirements (which refer to internal and external quality

attributes, respectively), and therefore more likely to meet user needs than 'creative' approaches.

The Model-Driven Engineering paradigm (MDE) (Kent, 2002; Bézivin, 2004) is already contributing to the achievement of these WE goals. MDE considers models as primary engineering artefacts throughout the engineering lifecycle, and regards the software life cycle as a chain of model transformations. All models in an MDE approach are formally described by means of meta-models. MDE technologies combine (1) *Domain Specific Modelling Languages* (DSML's), whose type systems formalize the application structure, behaviour and requirements within particular domains, and (2) *transformation engines and generators* that analyze certain aspects of models and synthesize various types of artefacts (Schmith, 2006). In this context, Model-Driven Architecture (MDA) (Mellor *et al*., 2004) has become the industrial large-scale application of the MDE principles around a set of OMG standards, such as MOF (OMG-MOF, 2006) to define meta-models, UML (OMG-UML, 2005) and OCL (OMG-OCL, 2006) to define models and XMI (OMG-XMI, 2005) to interchange models among tools.

The WE community has largely shifted to MDE, providing WE methodologies with the degree of formalization and process standardization that they lacked up to the advent of this paradigm. Meta-models have been defined, facilitating the understanding of the different model semantics (Koch and Kraus, 2003; Schauerhuber *et al*., 2006). Models have been reclassified as being Computation Independent Models (CIM), Platform Independent Models (PIM) or Platform Specific Models (PSM), depending on their position in the standard meta-pyramid of OMG (Assmann *et al*., 2006). Transformations between models, until now hardwired inside the different Computer Aided Web

Engineering (CAWE) development environments, have also been externalized (Koch 2006; Meliá *et al.*, 2005) by means of standard languages, typically QVT (OMG-QVT, 2005) or QVT-derived, such as the UML Profile Transformation (UPT) (Meliá and Gómez, 2006).

Resulting from these changes, the new MDE-based WE development process (see Figure 3) defines (1) a requirements workflow, whose outgoing artefact is a use case model, (2) an analysis workflow, whose output is a domain model (Entity Relationship or class diagram), (3) a conceptual design workflow, whose output is a navigational and a presentation model (expressed by means of UML profiles or proprietary notations), (4) a detailed design workflow that introduces platform and technology specific features (typically J2EE and .NET) and (5) an implementation workflow, which results in a Web application that is ready to be deployed. Variants of this process model exist, usually to include additional PIM and/or PSM models (architectural models, business process models, different languages and/or platforms, etc.) that further enrich the application specification. Also, a set of automatic or semi-automatic transformations among artefacts have been defined to streamline the process and to improve traceability among and between concepts.

This move to MDE has positioned the WE community closer than ever to the fulfilment of the first of the promises set out at the beginning of this section, that is, the achievement of a significant decrease in total costs and time-to-market of applications developed with WE methodologies. Unfortunately, the path towards the assurance of higher navigability of Web applications is not that well paved. Under the next heading

we analyse the state of the art in navigability measurement and introduce the topic of this paper, i.e. how an MDE approach can contribute to making advances in this field.

Early Navigability Measurement in WE

The navigability of a Web application in use, understood as the efficiency, effectiveness and satisfaction with which users can move around in the application to satisfy specific goals under specific conditions, is widely recognised as a milestone for the success of Web applications. This fact is reflected in the myriad of design guidelines (Nielsen, 2000) and automated measures (Ivory, 2004) that have been published.

While guidelines are, for most part, ambiguous and hard to follow (Ivory and Megraw, 2005), measures provide a systematic and accurate way of evaluating products. This fact is supported by empirical evidence: usability prediction of Web interfaces with the help of measures matches in some cases up to 80% of the results based on expert evaluation of the same Web pages (Ivory and Hearst, 2001).

Navigability assessment as usually carried out today suffers from two problems. The first one is that, more often than not, the navigability measurement process is based on heuristics that lack empirical validation (more than 60% of the heuristics are not validated (Calero *et al.*, 2004)). The second problem, which is where the focus of this paper lies, is that the measurement process is usually performed once the application has already been deployed, when errors and navigability problems are difficult and costly to remedy. Similar remarks also apply to Software Engineering, where research has concentrated mainly on the implementation (Auer, 1998) despite the fact that, as Fisher (1999) found, from a user's perspective an improved system results when technical communicators are involved particularly in the early stages of the development process.

Regarding web applications, the delay in measuring navigability is due to the fact that the degree of navigability perceived by the final user can only be directly assessed through the use of measures over external attributes under real conditions of use (ISO/IEC 9126, 2001).

Taking these considerations into account, our research framework is based on the conjecture that it is possible to establish a set of relationships between the navigability of a Web application in use and external/internal navigability properties of the Web application, as we can observe in Figure 1, which is based in the ISO 9126 standard for software quality (ISO/IEC 9126, 2001). External navigability (measured through external measures over external attributes) refers to the behaviour of the system during the testing and/or operational stages of the life cycle process. Internal navigability (measured through internal measures over internal attributes) is in turn assessed on the non-executable product during its development stage. Given that internal navigability influences external navigability, which for its part influences navigability in use, we can conclude that the latter can be at least partially assessed by taking measures from early navigability artefacts.

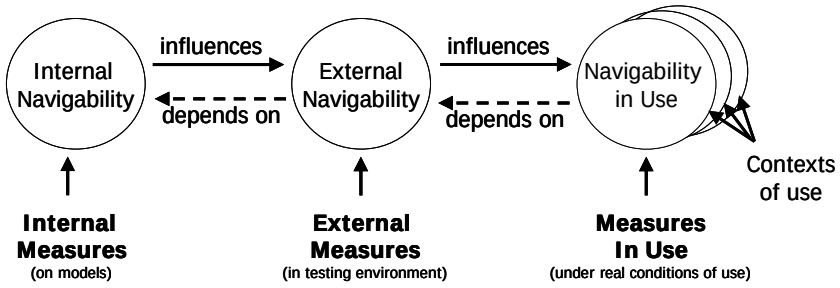

Figure 1: Relationship between navigability measures (adapted from ISO/IEC 9126)

This research framework is actually based on similar research in Empirical Software Engineering, where the relationship between measures of internal quality attributes and external quality attributes has been investigated (Briand and Wüst, 2002). Although

this stream of research is highly explorative in nature, consensus is emerging regarding the role that structural software properties (which can be assessed through internal measures taken from models that emerge at the software design stage) play in determining external software quality and the external software quality characteristics in the software quality in use (Bevan and Azuma, 1997). Reasons why structural properties have an impact on quality have been suggested by Briand *et al.* (1999b, 2001) (see **¡Error! No se encuentra el origen de la referencia.**). According to them, software that is big (i.e. having many composing elements) and has a complex structure (i.e. showing many relationships between the software's composing elements) results in a high cognitive complexity, which is defined as the mental burden of the people that perform tasks on the software. It is the high cognitive complexity that causes the software to display undesirable properties such as high effort to maintain, simply because it is more difficult to understand, develop, modify or test such software.

Briand *et al.*'s model (Briand et al, 1999b), which they refer to as a 'causal chain', has been the basis for much of the recent research on empirically-derived metrics-based software quality prediction (El-Emam et al., 2001; Genero et al., 2003, 2007; Poels and Dedene, 2001). The (indirect) relationship between software's structural properties and external quality properties has been repeatedly demonstrated. According to Briand *et al.* (2001), it is difficult to imagine what could be alternative explanations for these results besides cognitive complexity mediating the effect of structural properties on software quality.

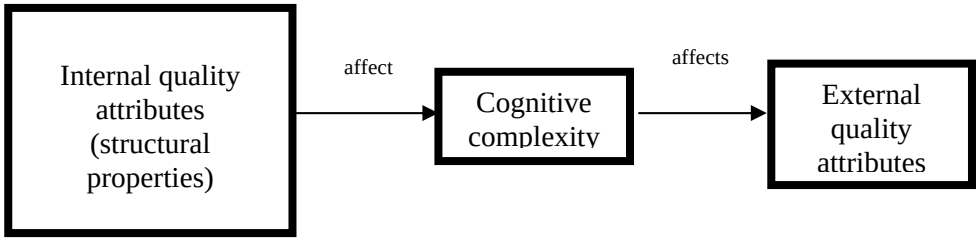

Figure 2. Relationship between internal quality attributes, cognitive complexity and external quality attributes (Briand et al. (1999b, 2001))

In the early assessment of structural navigation properties WE methodologies are called to play an important role. WE methodologies provide a software artefact that is specifically devoted to reflecting navigation design decisions: the Navigational Model (NM).

Therefore, we will use Briand *et al.*'s model combined with ISO 9126´s model as a working hypothesis and framework for our research: if it is assumed that the structural properties (internal navigability) of a NM affect its external navigability, then measuring these structural properties can be used as early indicators of external navigability and navigability in use. Moreover, the inclusion of measures that guide the construction of the NM may contribute to the assessment and improvement of internal navigability, with all the advantages that an early detection and correction of navigability problems implies (Briand *et al.*, 1999).

At this point we face a problem: measures taken on NMs are scarce in the literature (Abrahao *et al.*, 2003; Cachero, 2005; Comai *et al.*, 2002). Additionally, all the proposed measures are tightly coupled to the WE methodologies for which they were proposed. Unfortunately, these methodologies usually differ not only in notation but also in the semantics associated to some (although fortunately not all) of their constructs. This poses difficulties in the reuse of certain measures. Furthermore, only part of these measures have been defined in a formal way, and the effects that an out of bounds value may have on the models (that is, the modifications that should be performed on the design in order to improve the measure) are usually not specified.

Contribution of the paper

The main goal of this paper is to show how MDE principles and technologies can contribute to systematizing and automating an early navigability assessment and improvement process carried out on navigational models. In pursuit of this objective, section 2 gives an overview of related work in early navigability measurement. We first present the set of internal navigability measures that have been proposed in the literature for specific WE methodologies, even though their theoretical and empirical validation remains an open issue that is beyond the scope of this paper. We then present a table of navigational model construct equivalences that ensures the reusability of the (informal) measure definitions across four of the best known WE methodologies, namely OO-H (Gomez *et al.*, 2001), OOWS (Pastor *et al.*, 2001), UWE (Hennicker and Koch, 2000) and WebML (Ceri *et al.*, 2000). In section 3 we argue that the internal navigability measures that are relevant for a given application can be more rigorously defined by means of a new artefact: the *navigability measurement model*, which is obtained by instantiation of the Software Measurement Meta-model (SMM) that is obtained from the Software Measurement Ontology proposed by García *et al.* (2005). The use of this meta-model makes the model ripe for reuse within methodologies. This navigability measurement model can be integrated into the MDE-based WE process (see Figure 3). In this Figure, activities and artefacts related to navigability assessment and improvement are marked in green.

For this process to be complete, an endogenous mapping (Caplat and Sourrouille, 2003), involving the navigational meta-model provided by a given WE methodology, needs to be defined. For the sake of illustrating the approach, we have chosen OO-H. A brief introduction to this methodology, as well as to its navigational meta-model, is presented

in section 4. Section 5 takes an in-depth look at the details of the transformation that, expressed in UPT (Meliá and Gómez, 2006) allows the specification of a possible conversion of an OO-H navigational model into an OO-H navigability-improved navigational model by means of a UML profile. Our approach has been implemented as an extension of an existing MDE tool, the WebSA Tool (Meliá and Gómez, 2006) that is presented in section 6. Finally, in section 7 we present our conclusions and some further lines of research.

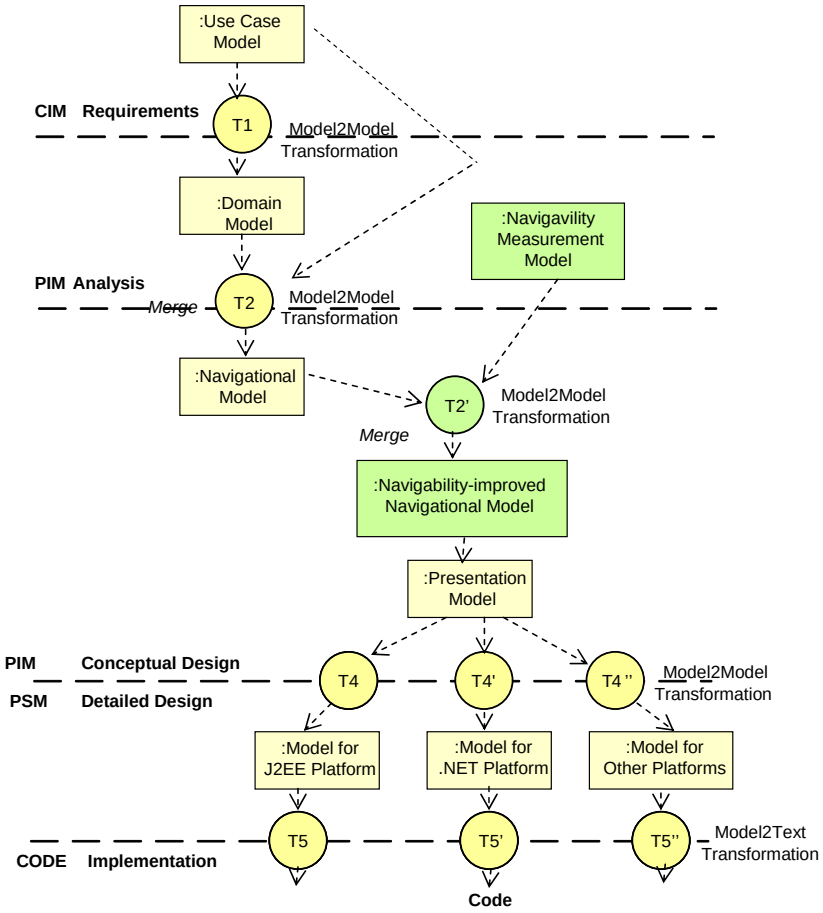

Figure 3: Navigability-aware MDE based WE Process

Related Work: Navigability Measures in WE

As we have stated above, internal navigability measures are few and far-between in the literature. Only WebML (Comai *et al.*, 2002), OOWS (Abrahao and Insfran, 2006; Abrahao *et al.*, 2003) and OOH (Cachero, 2005) tackle this issue explicitly. The

WebML Quality Analyzer (Comai *et al*., 2002) allows the analysis of the XML specification of WebML conceptual schemas. WebML defines two quality attributes that may contribute to assessing navigability based on the navigational model: *correctness* and *usability*. Syntactic correctness (assured by the methodology meta-model) is distinguished from semantic correctness. To get an indicator of semantic correctness, WebML proposes the measurement of the absence of conflicts, the absence of race conditions and the reachability of units and pages, but they do not provide concrete formulae to compute these indicators. Regarding usability, WebML distinguishes between consistency, ease of navigation and low page density, but again they only give pattern examples, and fail to define formally any measure that could be generally applicable to other WE navigational models. OOWS, focusing more on how the structural complexity of navigational models may affect quality in use attributes, does formally define a set of measures to assess the size, width, depth and edge-to-node ratio of OOWS navigational models. Based on this work, OO-H adapts such measures to OO-H, and puts forward a set of additional measures that can be used to assess consistency and ease of navigation in OO-H navigational models quantitatively, although these measures still need to be theoretically and empirically validated.

An overview of the measures proposed in OOWS and OO-H is presented in Table 1. This table does not include the measures of WebML, due to the lack of definitions. We have also left out of the table the measures that cannot be expressed in terms of the navigational constructs presented in Table 2, whose semantics are shared across methodologies.

Measure Name	**Measure Definition**
Number of Navigational Contexts (NNC) (Abrahao *et al.*	The total number of navigational contexts (context + sequence) in a

2003)	navigational map.
Number of Navigational Links (NNL) (Abrahao *et al.* 2003)	The total number of navigational links in a navigational map.
Density of a Navigational Map (DeNM) (Abrahao *et al,*. 2003)	Number of navigational links/Number of navigational contexts in a navigational map
Depth of a Navigational Map (DNM) (Abrahao *et al.*, 2003)	The longest distance from a root navigational context to a leaf context.
Breadth of a Navigational Map (BNM) (Abrahao *et al.*, 2003)	The total number of exploration navigational contexts in a navigational map.
Minimum Path Between Navigational Contexts (MPBNC) (Abrahao *et al.*, 2003)	The minimum number of navigational links that are necessary to cross over from a given source to a given target navigational context within a navigational map.
Navigation Pattern Coherence of a Navigational Model (NPCNM) (Cachero, 2005)	Degree of homogeneity in the application of the existing navigational patterns (index, guided tour, indexed guided tour, showall) to provide access to the system information.
Domain Coverage of the Navigational Model (DCNM) (Cachero, 2005)	*Percentage of domain relationships which, having already been defined as the conceptual relationships in which a certain user type is interested, can in actual fact be navigated by such a user.*
Navigational Model Simplicity (NMS) (Cachero, 2005)	Percentage of navigational links in a navigational model that support domain relationships.
Number of Paths Between Navigational Contexts (NPBNC) (Abrahao *et al.*, 2003)	The amount of alternative paths to cross over from a given source to a given target navigational context within a navigational map.
Compactness (Cp) (Abrahao *et al.*, 2003)	The degree of interconnectivity of a navigational map.

Table 1: Internal Navigability Measures

Perhaps the most important characteristic of these measures is that they can be reused among methodologies. However, this reuse is not evident, due to the already-mentioned divergence in navigational constructs and nomenclature that exists in the WE field. To overcome this problem, we have complemented Table 1 with a list of term equivalences found in four of the best known WE methodologies. This list (see

Table 2) includes all the navigation constructs involved in the definition of the measures presented in Table 1.

	OOWS	OO-H	UWE	WebML
Domain Model	*Object Model*	*Class diagram*	*Class Diagram*	*ER Model*
	Class	*Class*	*Class*	*Entity*
	Relationship (association, aggregation, composition)	*Relationship (association, aggregation, composition)*	*Relationship (association, aggregation, composition)*	*Relationship*
Navigation Model	*Navigational Map*	*Navigational Model*	*Navigational Model*	*Hypertext structure schema*
	Exploration Navigational Context	*Navigational Target*	*Package*	*Site View*
	Sequence Navigational Context	*Navigational Target*	*Package*	*Site View*
	Navigation Class	*Navigational Class*	*Navigation Node*	*Data Unit*
	Navigation Link	*Traversal Link Requirement Link Service Link*	*Navigation Link*	*Link*
	Navigation Patterns	*Navigation patterns*	*Access primitives*	*Sortable Unit*

Table 2: Equivalence of WE Navigational Constructs

With this table, translation to the various different methodologies of the measures presented in Table 1 is straightforward.

Given the facts that (1) the focus of the paper is the presentation of an MDA-based process that permits to automatically assess and improve structural navigability measures on early WE artifacts (navigational models), and not the validation of such measures; (2) the use of complex measures do not guarantee the success of the measurement; and (3) the scientific community has not yet provided an empirically validated set of measures that can be used to assess navigability, we have decided to use the relatively simple DCNM measure to illustrate our approach. This measure was

originally defined in terms of OO-H constructs and it is marked in bold in Table 1. Replicating the process presented in this paper with any empirically validated measure or indicator proven successful to actually improve the navigability in use of the application is straightforward, provided that such measures are based on data maintained in the origin and/or target meta-models. Being OCL the chosen language to express such measures and decision criteria, its expressive power assures that any kind of structural restriction on the underlying meta-models can be defined as part of the transformations.

The Domain Coverage of the Navigational Model (DCNM) measure

The DCNM measure was informally defined in (Cachero, 2005) as the *Percentage of domain relationships which, having already been defined as the conceptual relationships in which a certain user type is interested, can in actual fact be navigated by such a user.* The rationale of this measure lies in the assumption that users may expect to find in the Web application the same relationships that exist among concepts in the problem space. Not finding these relationships in the application may therefore diminish their general satisfaction with the application. Users are likely to describe this phenomenon as a problem with the navigability of the application.

If we wish to obtain a value for this measure in OO-H, the first step consists in checking which *domain classes* in the *class diagram* contribute to making up the user view, that is, which of them support the *navigational classes*. We must then count the Number of Relevant Relationships (NRR) that exists in the *class diagram* among such *domain classes*. The reason why we are just interested in the NRR subset is that not every concept is relevant for every *navigational model*. The third step consists in counting the relationships that support navigation in the *navigational model* (i.e. counting the

traversal links). This measure is referred to as the Number of Navigated Domain Relationships (NNDR). With these two measures it is possible to calculate the DCNM measurement result as NNDR/NRR*100. In this definition, words in italics refer to OO-H specific concepts that will be further explained in section 4.

Let's assume now that we want to redefine the calculation of this measure in terms of a different WE methodology. For the sake of the example, we have chosen WebML, which, not being originally based in UML, differs most in terminology with respect to OO-H. Applying the list of equivalences presented in Table 2, the first task is to check which *entities* in the *ER model* contribute to making up the user view, that is, which of them support the *data units*. Secondly, we must count the Number of Relevant Relationships (NRR) existing in the *ER model* among such *entities*. In the third place, the *links* that appear in the *hypertext structure schema* must be counted. This value is referred to as the Number of Navigated Domain *Relationships* (NNDR). With these two values it is again possible to calculate the DCNM measurement result with the measurement function NNDR/NRR*100.

The main problem with these informal definitions lies in the fact that they may be hard to understand for people trying to adopt the measures. Our next point is to present how this problem can be palliated by instantiating a common Software Measurement Meta-Model that avoids ambiguities and fosters the completeness of the measure definition.

Use of the Software Measurement Meta-Model for the Definition of Measures over Navigational Models

Although software measurement plays an increasingly important role in Software Engineering (Fenton and Pfleeger, 1997), there is no consensus on many of the concepts and terminology used in this field, such as, ‘measurement’, ‘measure’, ‘metric’, ‘measurable attribute’, etc. Worse still, vocabulary conflicts and inconsistencies are frequently found amongst the many sources and references commonly used by software measurement researchers and practitioners, including international Software Engineering standards (García *et al.*, 2005).

In an effort towards the harmonization of the different software measurement standards and research proposals, García *et al.* (2005) have proposed a Software Measure Ontology (SMO). This ontology provides a common conceptualisation of software measurement, where objects, concepts, entities and their relationships are explicitly represented in an unambiguous and explicit way. Setting as our objective the obtaining of a generic measurement model for navigational models, we have adopted SMO and its corresponding Software Measurement Meta-Model (SMM) (Ferreira *et al.*, 2006) (see Figure 4).

SMO and SMM are represented by using a UML diagram. UML-based ontologies and meta-models have the obvious advantages of being more widely understandable than ontological languages and of being aligned with the MDE (*Model Driven Engineering*) movement (PlanetMDE, 2005).

The classes in the SMM are packaged (shown using colours in Figure 4) according to the following sub-ontologies of the SMO:

- *Characterization and Objectives* includes the concepts required to establish the scope and objectives of the measurement process.
- *Software Measures* aims at establishing and clarifying the key elements in the definition of a measure.
- *Measurement Approaches* introduces the concept of measurement approach, to generalize the different 'approaches' used by different kinds of measures for obtaining their respective measurement results.
- *Measurement Action* establishes the terminology related to the act of measuring.

The "Software Measurement Characterization and Objectives" sub-ontology includes the concepts required to establish the scope and objectives of the software measurement process. The main goal of a software measurement process is to satisfy certain information needs by identifying the entities (which belong to entity classes) and the attributes of these entities (which are the focus of the measurement process). Attributes and information needs are related through measurable concepts (which belong to a quality model).

The "Software Measures" sub-ontology aims at establishing and clarifying the key elements in the definition of a software measure. A measure relates a defined measurement approach and a measurement scale (which belongs to a type of scale). Most measures may or may not be expressed in a unit of measurement (for example, nominal measures cannot be expressed in units of measurement), and can be defined for more than one attribute. Three kinds of measures are distinguished: base measures, derived measures, and indicators.

The “Measurement Approaches” sub-ontology introduces the concept of measurement approach to generalize the different “approaches" used by the three kinds of measures for obtaining their respective measurement results. A base measure applies a measurement method. A derived measure uses a measurement function (which rests upon other base and/or derived measures). Finally, an indicator uses an analysis model (based on a decision criteria) to obtain a measurement result that satisfies an information need.

The “Measurement” sub-ontology establishes the terminology related to the act of measuring software. A measurement (which is an action) is a set of operations having the object of determining the value of a measurement result, for a given attribute of an entity, using a measurement approach. Measurement results are obtained as the result of performing measurements (actions).

Table 3 shows the concepts defined in the ontology.

Term	**Definition**
Measurement Approach	Sequence of operations aimed at determining the value of a measurement result. (A measurement approach is either a measurement method, a measurement function or an analysis model)
Measurement	A set of operations having the object of determining the value of a measurement result, for a given attribute of an entity, using a measurement approach
Measurement Result	The number or category assigned to an attribute of an entity by making a measurement
Information Need	Insight necessary to manage objectives, goals, risks, and problems
Measurable Concept	Abstract relationship between attributes of entities and information needs
Entity	Object that is to be characterized by measuring its attributes
Entity Class	The collection of all entities that satisfy a given predicate
Attribute	A measurable physical or abstract property of an entity, that is shared by all the entities of an entity class
Quality Model	The set of measurable concepts and the relationships between them which provide the basis for specifying quality requirements and evaluating the quality of the entities of a given entity class
Measure	The defined measurement approach and the measurement scale. (A measurement approach is either a measurement method, a measurement function or an analysis model)

Scale	A set of values with defined properties
Type of Scale	The nature of the relationship between values on the scale
Unit of Measurement	Particular quantity, defined and adopted by convention, with which other quantities of the same kind are compared in order to express their magnitude relative to that quantity
Base Measure	A measure of an attribute that does not depend upon any other measure, and whose measurement approach is a measurement method
Derived Measure	A measure that is derived from other base or derived measures, using a measurement function as measurement approach
Indicator	A measure that is derived from other measures using an analysis model as measurement approach
Measurement Method	Logical sequence of operations, described generically, used in quantifying an attribute with respect to a specified scale. (A measurement method is the measurement approach that defines a base measure)
Measurement Function	An algorithm or calculation performed to combine two or more base or derived measures. (A measurement function is the measurement approach that defines a derived measure)
Analysis Model	Algorithm or calculation combining one or more measures with associated decision criteria. (An analysis model is the measurement approach that defines an indicator)
Decision Criteria	Thresholds, targets, or patterns used to determine the need for action or further investigation, or to describe the level of confidence in a given result

Table 3. SMO terms definition

For a better understanding of the SMO and SMM, interested readers are referred to (García *et al.*, 2005) where an exhaustive definition of the concepts defined in the four packages can be found.

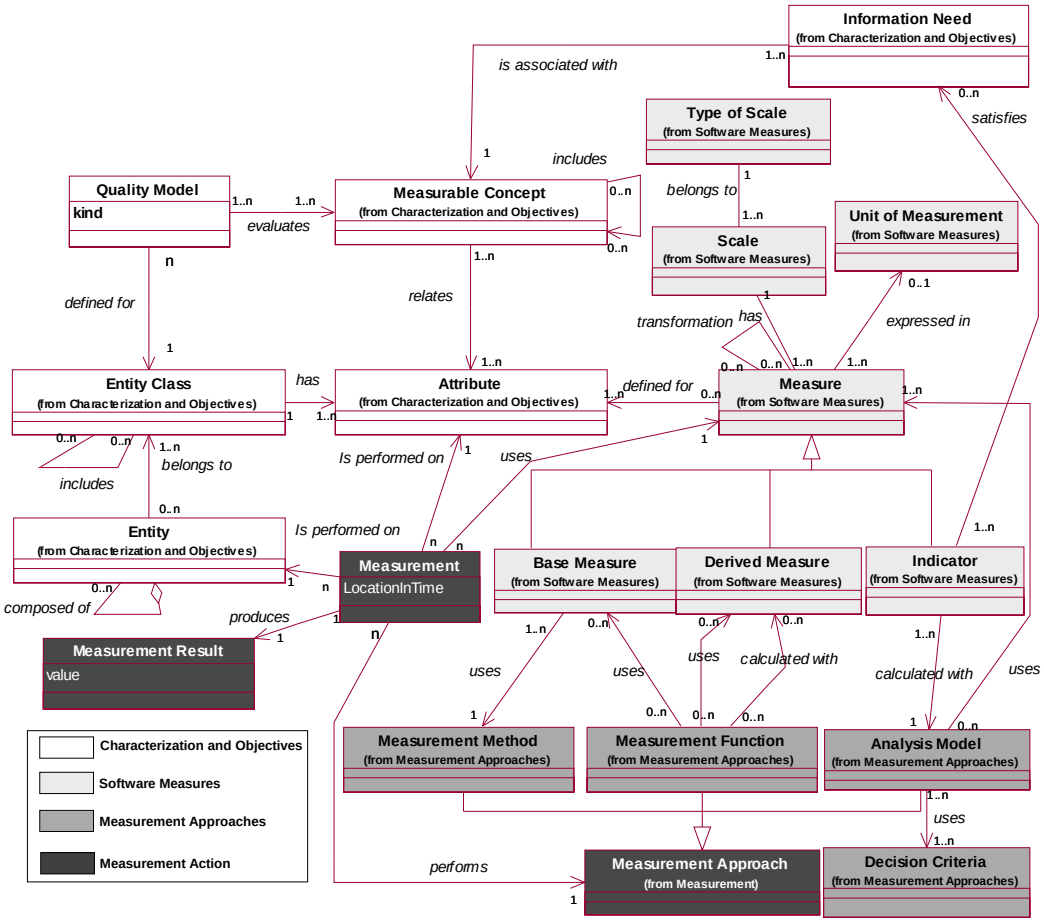

Figure 4: The Software Measurement Metamodel (SMM)

Our navigability measurement meta-model is defined, therefore, as an instantiation of the SMM. We now go on to illustrate the instantiation of SMM for defining a concrete navigability measurement model that, for the sake of simplicity, only covers the DCNM measure, which was informally introduced in section 2.

SMM instantiation for the DCNM measure

Figure 5 shows the instantiation of SMM for defining, in an unambiguous way, a navigability measurement model that includes the *DCNM* **Measure** through an object model. This **Measure** aims at providing useful information to satisfy the **Information Need** "Knowing the Navigability of a Navigational Model of a Web application" (*To*

Know Navigability). The **Measure** is defined in the context of a **Quality Model**, named *NM* (Navigability Model). This **Quality Model** is defined for the **Entity Class** *Web Conceptual Model* (an abstract model that comprises all the PIMs associated with a web application), which in our example includes two Entity Classes: *Navigational Model* and *Domain Model*. The NM evaluates the **Measurable Concept** *Navigability*.

The **Measurable Concept** *Navigability* is related to three **Attributes:** *Domain Coverage*, *Structural Complexity NM and Structural Complexity DM*. The *Navigational Model* has two **Attributes:** *Domain Coverage* and *Structural Complexity NM.* The Domain Model has the **Attribute** *Structural Complexity DM.*

The **Derived Measure** *DCNM* is defined for the **Attribute** *Domain Coverage.* DCNM is calculated with a **Measurement Function** (*DCNMFunction*), which uses two **Base Measures**, *NNDR* and *NRR*, defined in OO-H as follows:

- *NNDR*: Number of Navigated Domain Relationships that are related to traversal links in the Navigational Model.
- *NRR*: Number of Relevant Relationships between classes in the Domain Model (these classes should be the same classes that appear in the *Navigational Model*).

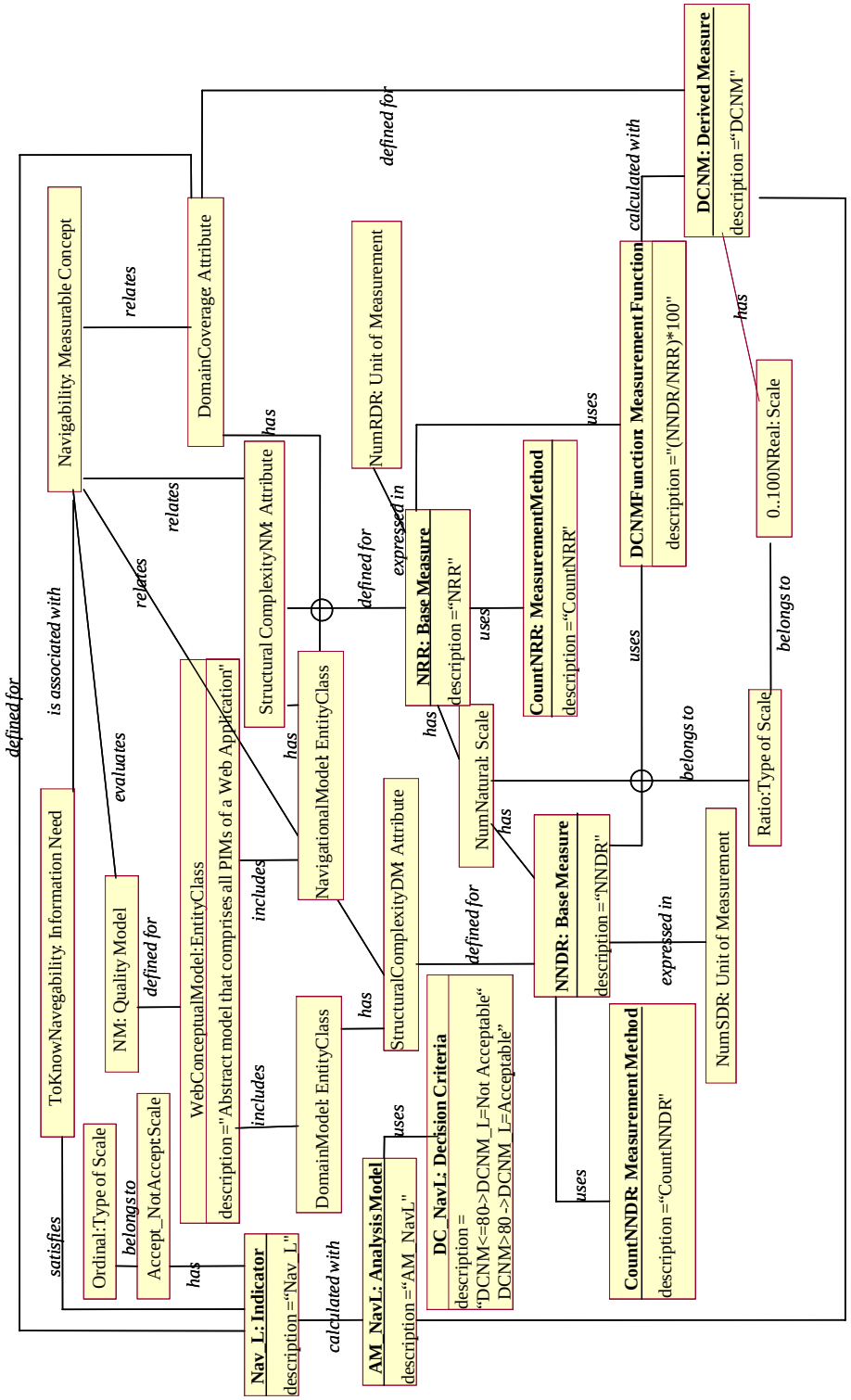

Figure 5: A Navigability Measurement Model

DCNMFunction is then defined as follows: (NNDR/NRR)*100.

NNDR is defined for the **Attribute** *Structural Complexity NM*. *NRR* is defined for the **Attribute** *Structural Complexity DM*. Furthermore, both *NNDR* and *NRR* have the **Scale** *Natural Number* (*NumNatural*) which belongs to the **Type of Scale** *Ratio*. The **Unit of Measurement** of *NNDR* is *numSDR* (number of supported domain relationships). The **Unit of Measurement** of *NRR* is *numRDR* (number of relevant domain relationships).

NNDR is calculated by a **Measurement Method** *CountNNDR*, which counts the total number of supporting relationships in a *Navigational Model*. *NRR* is calculated by a **Measurement Method** *CountNRR*, which counts the total number of relevant relationships between classes in the *Domain Model* (these classes should support the classes that appear in the *Navigational Model).*

DCNM is a percentage, so it has the **Scale** *0..100NReal* (real numbers in the range 0..100), which belongs to the **Type of Scale** *Ratio*. This is an scalar value, so it does not have **Unit of Measurement.**

For the satisfaction of the **Information Need** we have defined the **Indicator** *Navigability Level (Nav_L). Nav_L* is calculated with an **Analysis Model**, named *AM_NavL*, which uses the measure DCNM.

AM_NavL also uses a **Decision Criterion**, given the name of *DC_NavL*, for the calculation of the **Indicator** *Nav_L*. *DC_NavL*, which is defined as follows:

DCNM<=80 → *Nav_L*= Not Acceptable

DCNM>80→ *Nav_L*= Acceptable

So the **Indicator** *Nav_L* can take two values, Acceptable or Not Acceptable. This **Indicator**, as a measure, has a discrete **Scale** *Accept_NotAccept*, which belongs to the **Type of Scale** *Ordinal*.

In Table 4 we show the equivalences among the example instantiations and the navigability measurement meta-model terms.

Term	**Instantiation for the Navigability Measurable Concept**
Information Need	Knowing the Navigability of a Navigational Model of a Web application
Measurable Concept	Navigability
Entity Class	Web conceptual model: navigational model and domain model
Attribute	(1) Domain coverage – (2) Structural complexity NM (for NM) – (3) Structural complexity DM (for DM)
Quality Model	NM-Navigability Model
Derived Measure	DCNM
Measurement function	DCNMfunction=(NNDR*NRR)/100
Scale	0..100NReal
Type of Scale	Ratio
Unit of Measurement	-
Base Measure	(1) NNDR - (2) NRR
Scale	Natural Number
Type of Scale	Ratio
Unit of Measurement	(1) numSDR- (2)numRDR
Measurement Method	(1) Count the total number of supporting relationships in a Navigational Model– (2) Count the total number of relevant relationships between classes in the Domain Model
Indicator	Nav_L
Scale	0Acceptable-NonAcceptable
Type of Scale	Ordinal
Analysis Model	Am_NavL=f(DCNM)
Decision Criteria	If f(DCNM) <=80 then NonAcceptable else NonAcceptable

Table 4. WE-Measurement Meta-model instantiation example

The resulting DCNM measurement model is reusable across WE methodologies. Only the details of the measurement methods for the base measures (i.e. the particular way of

counting relationships and links, encapsulated in the *CountNNDR* and *CountNRR* concepts) need to be adapted to the constructs of the chosen WE methodology.

This adaptation can be again facilitated by the set of equivalences presented in Table 2 (see also the example given above, where DCNM was redefined in terms of WebML). Furthermore, since the measurement model is independent of any concept regarding the particular application domain, we can reuse the navigability measurement model across application domains. We must, however, be careful with this reuse, as empirical evidence indicates that acceptable values of measures (which are included in the navigability measurement model by means of the *decision criteria* instantiation) may differ among Web application domains (e.g. e-commerce, educational, financial, etc.) (Ivory and Hearst, 2001).

In the following section we show how the navigability measurement model presented in Figure 5 can be integrated into an MDE-based WE development process (see Figure 3) to facilitate the automatic assessment and improvement of measures performed on WE navigational models. This integration can only be achieved when based on an underlying navigational meta-model. At this point we face a problem: the WE field has not yet reached an agreement on a common set of navigation concepts that could make up a general meta-model. For this reason, the definition of the transformations needed to assess and evolve the navigational model according to the measures included in the navigability measurement model must be specifically defined for each WE methodology. To illustrate our approach, we have chosen OO-H, whose fundamentals are presented next.

OO-H in a nutshell

OO-H is a WE methodology that provides the designer with the process and notation necessary for the development of Web-based interfaces and their connection with previously-existing business logic modules.

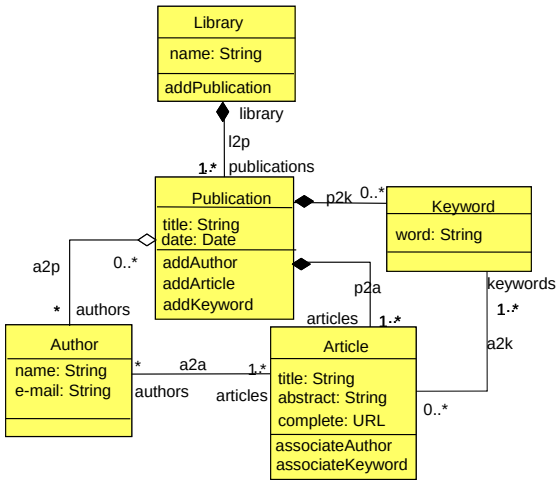

Figure 6: Library System Domain Model

To achieve this goal, OO-H provides three complementary views: a domain model, which is expressed by means of a UML-compliant class diagram, a navigational model, based on the domain model, and a presentation model, which describes the appearance of individual Web pages. The methodology is supported by a CAWE tool called VisualWADE (Cachero, 2003), that includes a model compiler capable of generating the data sources and the logic modules in the desired implementation environment. The syntactic correctness of the OO-H models is achieved through the definition of an underlying, MOF-based meta-model, which systematically and unambiguously defines the concepts involved and their relationships. For the sake of simplicity, in this paper we will focus on the domain and navigational models, which are enough to illustrate our approach.

The OO-H Domain Model

As we have already stated, the OO-H domain model is defined by means of a UML class diagram. By way of example, let's suppose we want to model a Web application to deal with a *Library* system. The Domain model of such system is depicted in Figure 6. There, we can observe how the library keeps track of a set of *publications*, each one having a set of *keywords* by which it is characterized. Each publication is composed of a set of *articles*. Each article has a *title*, an *abstract* and an *URL* to the complete text. The articles are authored by zero or more *authors*, and are defined by one or more keywords among those defined for the publication in which they are contained. This model is the basis on which the navigational model is constructed, as we now go on to demonstrate.

The OO-H Navigational Model

The OO-H navigational model reflects the paths the user can follow through the information domain in order to achieve her goals.

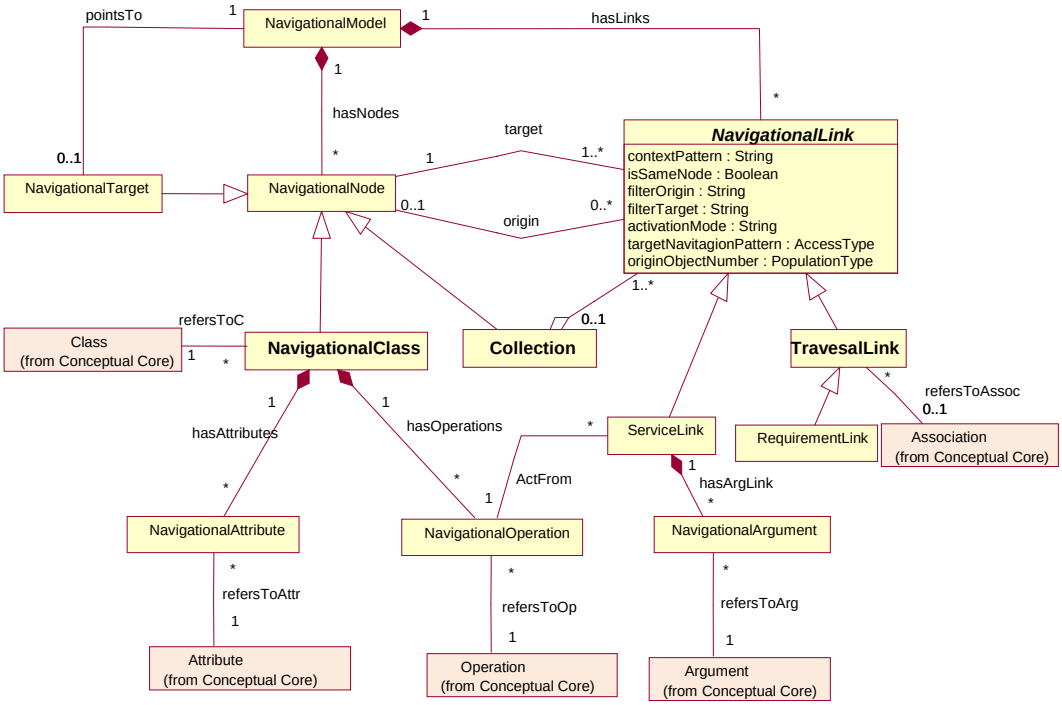

Figure 7: OO-H Navigational Meta-Model

It is usually organized around navigational targets (subsystems), each one encapsulating views defined over domain classes, as well as links used to interconnect them. This model is based on an OO-H navigational meta-model that is presented in Figure 7. In this Figure we can observe how, as in most WE approaches, the main constructs of the OO-H navigational model are *Navigational Classes* (NC), *Navigational Links* (NL) and *Collections* (C). Navigational Classes provide restricted views over conceptual classes. They define which items (attributes and operations), out of the set that the conceptual class provides, are accessible to the role that is associated with the Navigational Model. Navigational Links reflect navigation steps through the information. This is the richest construct in OO-H, which in this sense greatly differs from other approaches where the main construct is the Navigational Class. The attributes of the NL meta-class allow for the specification of not only the population of the target view (*targetFilter*) and the navigation structure through this population (*targetNavigationPattern*), but also the objects from which such navigation is possible (*originFilter*), the cardinality of the origin set of objects (*numberObjectsInOrigin*) and whether the user interacts or not with the application in order to activate such navigation (*activationMode*). In OO-H there are different types of NL: *Traversal Links* define navigation paths between *Navigational Nodes* (collections, classes and/or navigational targets), *Requirement Links* define starting points for user navigation, and *Service Links* capture navigation paths by which the user can interact with the underlying business logic. Within these, traversal links are the most commonly used, and may support the semantics of an underlying domain relationship. Lastly, *Collections* are access mechanisms (menus) that group together navigation paths. To illustrate their use, let's imagine that we want to model the navigation view of a reader actor who interacts with the library system depicted in

Figure 6. A navigational model example corresponding to this view is presented in Figure 8.

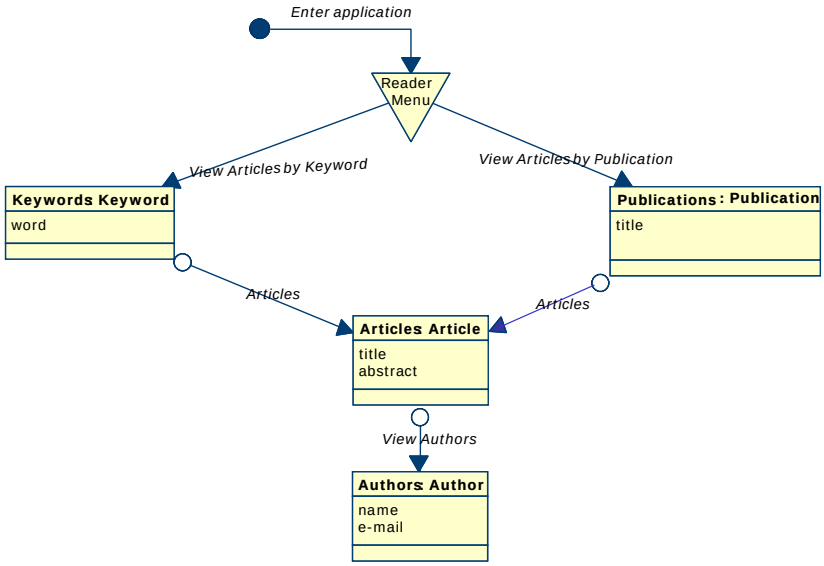

Figure 8: Initial Navigational Model Corresponding to a Reader Actor

This model reflects a system whose navigation is organized around a collection that, depicted as an inverted triangle, represents the *Reader Menu*. This menu allows the user to decide between two paths: *View Articles by Keyword* and *View Articles by Publication*. Both paths are defined by means of traversal links, which also serve to connect the remaining navigational classes that the model is composed of: *Keywords*, *Publications*, *Articles* and *Authors*.

These navigational classes are in fact in OO-H views of the corresponding UML domain classes. For example, *Publications* is a view of the domain class Publication, where all the operations and the *date* attribute have been hidden. As we have stated above, traversal links may reflect underlying conceptual relationships of the application domain. If this is the case, OO-H decorates the traversal link icon (an arrow) with an additional circle. In our example, the traversal link *'Articles'* defined between *Keywords* and *Articles* is an example of a traversal link that has been defined upon the underlying

conceptual relationship *a2k* that was presented in Figure 6. To further understand the navigational model depicted in Figure 8, a storyboard of the resulting application is presented in Figure 9.

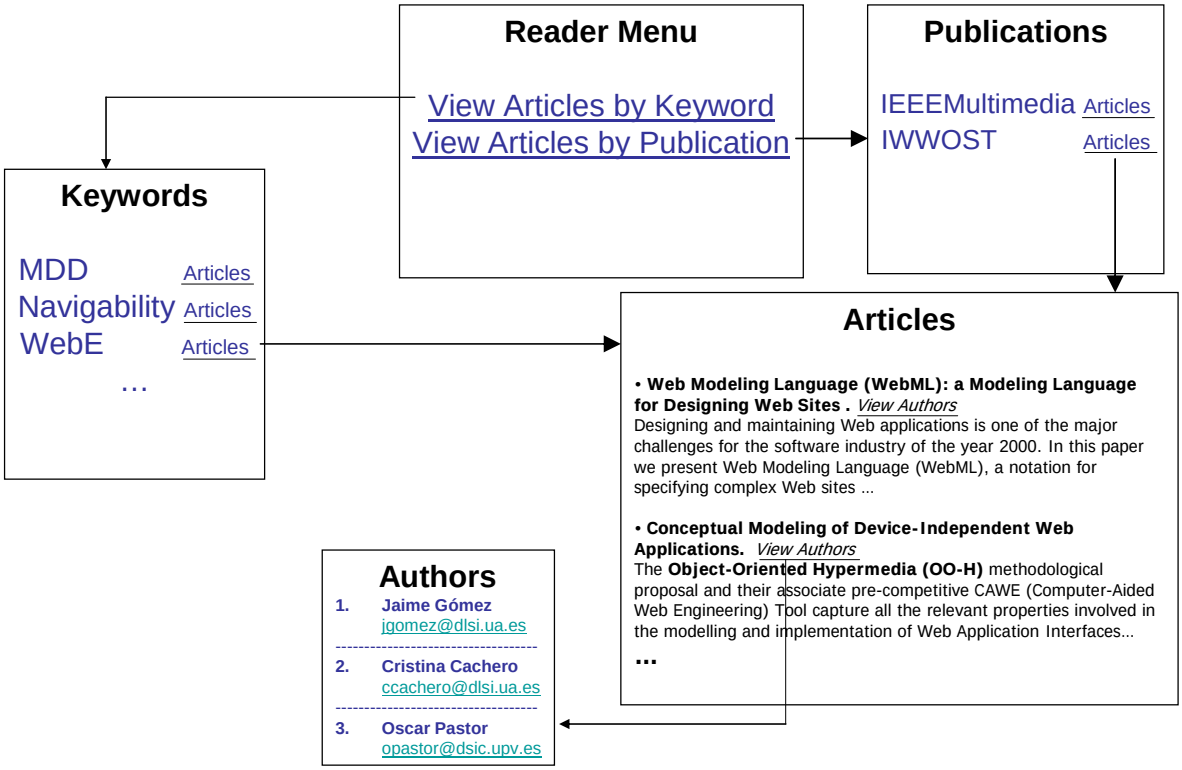

Figure 9: Storyboard of the Reader Actor Interface in the Library System

Let's now calculate intuitively the DCNM measurement result if we apply the measure to the OO-H navigational model depicted in Figure 8. Comparing Figure 6 and Figure 8, we can observe that there are four classes that support the navigational model: *Keyword*, *Publication*, *Article* and *Author*. These classes are related by means of two associations, one shared aggregation and two compositions, which makes a total of five relationships. Now, if we look at the navigational model, we can see three traversal links with a hollow circle at one of its ends, which means that only three out of the five relationships are supported by them. If we now apply the DCNM measurement function (3/5*100), we get a result of 60, which means that 60% of the domain relationships that could be navigated by the reader are in fact supported by her navigational model. This value

violates the level of acceptance for the Indicator *Nav_L*, because it is less than 80 (see Figure 5). Therefore, the navigational model must be improved by introducing certain design modifications (in this case the addition of new traversal links).

Our next intention is to set out how both the navigability assessment (e.g. measurement of the domain coverage attribute and subsequent calculation of the *Nav_L* indicator) and the navigational model evolution (e.g. model transformation to improve navigability) can be automated in the context of our MDE-based WE development process presented in Figure 3.

MDE Evaluation & Evolution of Navigational Models

MDE makes it possible to formalize the evaluation and evolution of the navigational models by means of artefacts called transformations. In fact, MDE transformations are themselves models, instances of a well defined meta-model and with the same reuse capabilities as any other model in the approach. Another important aspect is that these transformations clearly state the interconnection and traceability between concepts appearing in different models throughout the process (Balasubramanian *et al*., 2006).

Among the set of transformation languages proposed to define these transformations, the Query/Views/Transformations language (OMG-QVT, 2005), defined by the OMG, has raised particular interest within the research community. QVT suffers from several well known problems, however:

- The QVT Relations graphical language is based on a proprietary notation that (1) has to be learnt (higher learning curve) and (2) needs to be explicitly added to tools. This fact means that, up to now there have been no commercial graphical tools that support the QVT notation in full.

- The QVT Relations graphical language represents patterns at the object level. For this reason the graphical language lacks the expressivity that is needed to capture some meta-model characteristics (e.g. cardinalities), that can only be reflected at type level.

All these problems have led the research community to look for alternatives. One such alternative is UPT (Meliá and Gomez, 2006). UPT is a transformation language inspired by the QVT standard, which defines a MOF-compliant meta-model and a UML profile. This profile extends the UML class diagram semantics, and allows for (1) a lower learning curve to modelling transformations and (2) the use of UML tools to support the modelling process. These reasons have made us opt to define the transformation that automates the DCNM measure in UPT. Figure 10 shows how this transformation is defined as an instance of the UPT Meta-model. Furthermore, Figure 10 also presents the elements that take part in the definition of this transformation.

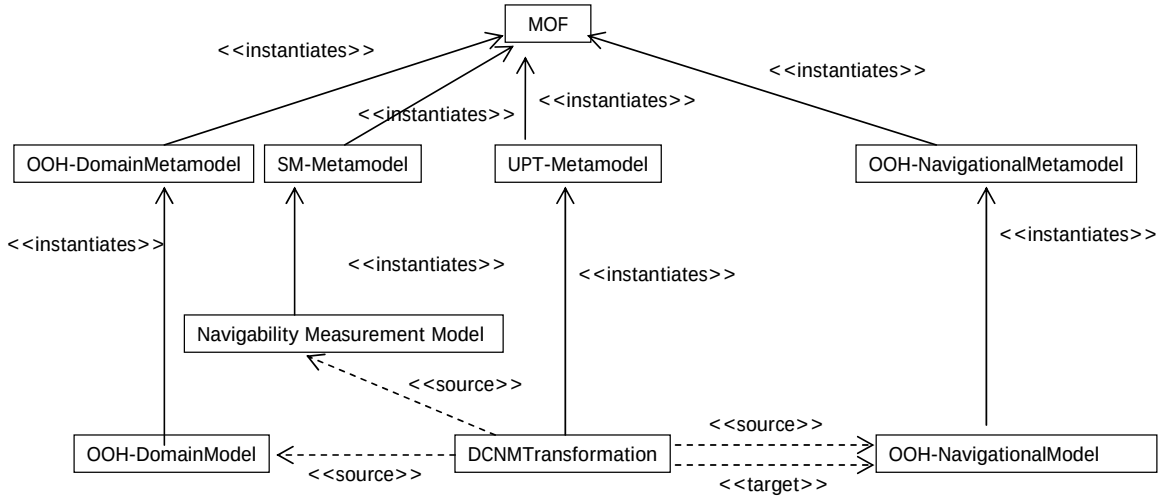

Figure 10: Model Transformation Pattern (adapted from Bézivin 2004)

The *DCNMTransformation* requires three source domains (that is, domains used to check the fulfilment of certain conditions) and one target domain (where modifications are performed if necessary). The first source domain is the *Navigability Measurement Model* (which contains the definition of the DCNM measure that is used in our example

to assess and improve the navigational model, see Figure 5). The second source domain is the *OO-H-DomainModel*, where the transformation checks the domain relationships that are relevant for the navigational model. The third source domain is the *OO-H-NavigationalModel*, where the transformation checks which relationships can already be navigated by the user. Lastly, the target domain is again the same *OO-H Navigational Model*, where new traversal links are included if needed. The following section shows how the DCNMTransformation manipulates each model.

DCNM Transformation Map

Model-driven navigability improvement of navigational models implies two main tasks: assessment and transformation. When aiming to simplify their materialization, it is usual to apply a divide-and-conquer strategy which consists in defining a chain of relations (also called rules) that makes up a transformation map. Going back to our example, Figure 11 shows the transformation map that depicts the set of rules that the *DCNMTransformation* is composed of. This transformation automates both the DCNM measurement and the navigational model redesign actions to be performed if the measurement result is out of the established bounds. Within this map, each task is performed by a different type of rule. On the one hand, the calculus of the measurement result is performed by means of a root *evaluation rule (*identified by the tagged values *relationType=evaluation, isRoot=true).* Evaluation rules are relations that do not perform changes on the models. They therefore work exclusively with *Check-only* domains (stereotyped <<C>>). Check-only domains are non-modifiable domains used by the transformation rule to check the compliance of certain conditions. On the other hand, the design decisions aimed at improving the measurement result are captured by *evolution rules*. Evolution rules work with both Check-only and *Enforceable* domains (stereotyped <<E>>). Enforceable domains are modifiable domains by which the

transformation declares the obligation for certain relationships and/or objects to be present/missing in the domain, causing the creation or deletion of model elements if necessary.

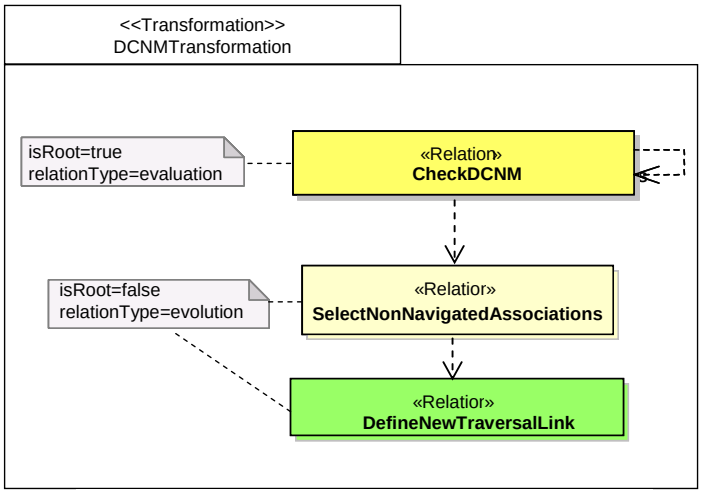

Figure 11: DCNM Transformation Map

We shall now examine each of these rules in detail.

DCNM Evaluation Rule: CheckDCNM

Figure 12 presents the *CheckDCNM* evaluation rule.

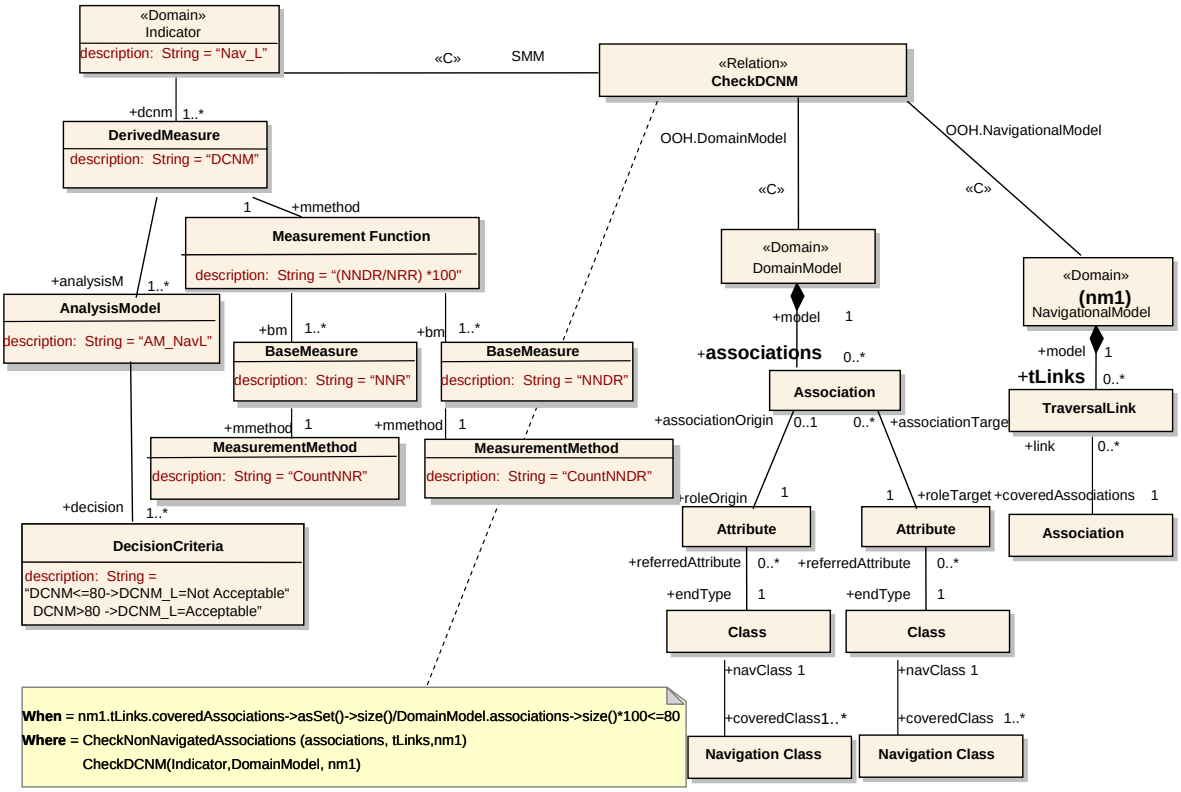

Figure 12: CheckDCNM Evaluation Rule

This relation must check whether the DCNM derived measure is applicable to the actual navigational model. This is done in the rule branch that starts with the *Indicator* Check-only domain. In Figure 12 we observe how each component of the derived measure pattern defined in this rule can be matched with a corresponding element in the model presented in Figure 5 (where, for the sake of readability, these have been marked in bold). This rule must, moreover, calculate the *Nav_L* measurement result, which depends only on the *DCNM* measure. We have already stated that, to get the *DCNMFunction* measurement function result, we needed to know two base measures: *NNDR* (number of navigated domain relationships) and *NRR* (number of relevant relationships). NNDR is calculated by means of the rule branch whose root is the check-only OO-H *NavigationalModel* domain (aliased *nm1*). This rule branch implements the *CountNNDR* measurement method. In this rule branch we have selected the traversal links that are related to a domain association (multiplicity 1). The number of traversal links that fulfil this condition is calculated by means of the OCL sub-expression nm1.tLinks.coveredAssociations->asSet()->size(), where the asSet() operation is an OCL operation that eliminates duplicates. This sub-expression is located in the *When* clause of the rule. The value of this sub-expression on Figure 8 would be three, which corresponds to the three links marked with a circle. NNR is in turn calculated by means of the rule branch whose root is the check-only OO-H *DomainModel* domain. In this branch (implementing the *CountNRR* measurement method), we have selected the associations that connect two classes that support at least one navigational class each (multiplicity 1..*). Again, the number of relationships (associations, aggregations, compositions) that fulfil this condition is obtained by means of the OCL sub-expression DomainModel.associations->size(), located in the When clause of the rule. The value of this sub-expression, when applied to the model presented in Figure 8, would be five,

which corresponds to the five relationships (two associations, one shared aggregation and two compositions) that exist between and among the four classes (Keyword, Article, Publication, Author) that take part in the support of the navigational model.

In addition, the *When* clause is responsible for the calculation of the DCNM measurement result. This result is calculated by dividing the two OCL sub-expressions and multiplying the result by 100. In our example, this operation gives a result of 60 (3/5*100).

Finally, the *Nav_L* measurement result (which takes the value of the DCNM measurement result) must be compared with the value established in the *DC_NavL* decision criteria (<=80). If the comparison is positive (which is the case for the navigational model of Figure 8), then the *Where* clause is executed. The Where clause includes two rule calls. The first call (SelectNonNavigatedAssociations) starts the chain of evolution rules that modify the navigational model *nm1* to improve the DCNM measurement result. This rule, as we have stated before, is called with three parameters (marked in bold in Figure 12): (1) the set of candidate associations (the five relationships in our example), (2) the set of traversal links (three in our example) that support an underlying domain relationship and (3) the navigational model (nm1 in our example) where the new traversal links are to be added. The second call is a recursive one that serves to check whether the measurement result is now acceptable once the model has evolved.

The fact that UPT is semantically richer than the QVT-Relations language makes possible to automatically derive the QVT-Relations textual specification out of the UPT

rule. The QVT-Relation specification of the three rules included in this paper is presented in Appendix A.

The SelectNonNavigatedAssociations Evolution Rule

The *SelectNonNavigatedAssociations* rule (see Figure 13) is in charge of selecting, from the set of relevant relationships, a random relationship among those that are not yet supported by the navigational model. The fact that the DCNM measurement result is lower than 100 ensures that there is at least one relationship that fulfils this restriction. To reach its objective, the rule compares the two sets of objects received as parameters, trying to find a domain relationship that does not correspond with any of the relationships supported by a traversal link in the second set. This condition is expressed in the When clause of the rule by the OCL expression *Not((nA=n1 and nB=n2) or (nA=n2 and nB=n1))*, where *n1* and *n2* are variables that refer to domain classes that are related by an association, and *nA* and *nB* are variables that refer to domain classes that support the navigational classes in the navigational model and which are connected by a traversal link. This expression also takes into account that bidirectional associations may be represented by traversal links going in any of the two directions.

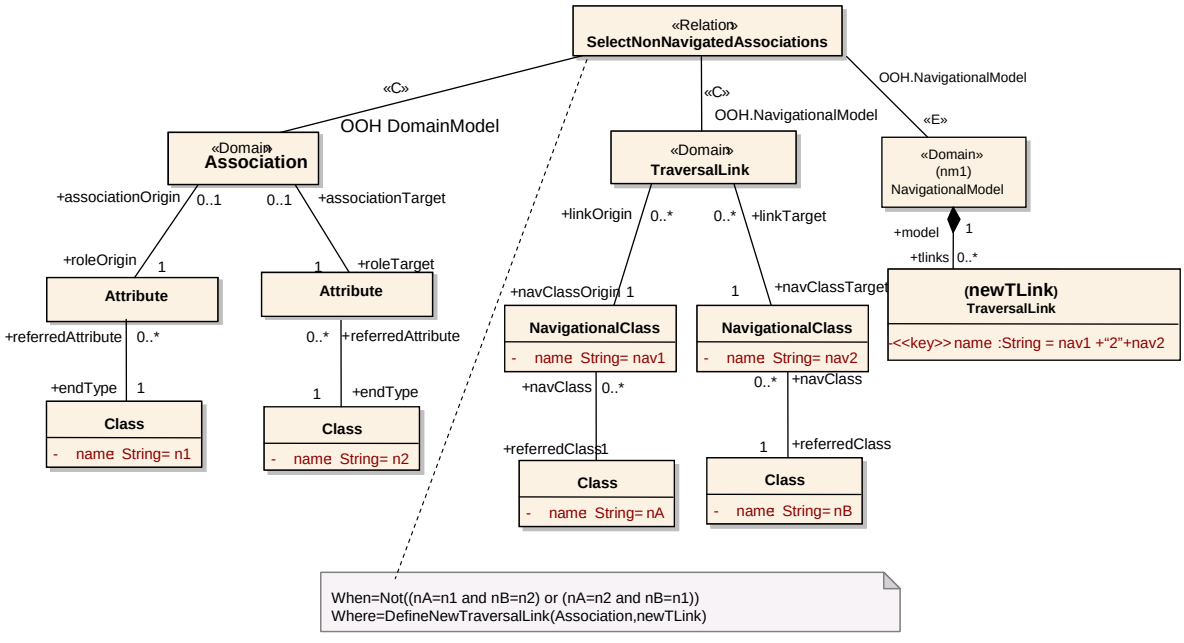

Figure 13: SelectNonNavigatedAssociations Evolution

The third branch of the rule corresponds to an enforceable domain with a *newTLink* element whose name is constructed by concatenation of the strings nav1+“2”+nav2. This name is decorated with a <<*key*>> stereotype. This stereotype causes the branch to look for any model element that matches the key value, and only if this is not found is a new element created. In our example (see Figure 8) we will suppose the rule has chosen the *p2k* relationship in the direction from the navigational class *Publications* to the navigational class *Keywords*. This relationship causes the name of the *newTLink* to be set to ‘Publications2Keywords’. The fact that the relationship is not supported in the navigational model ensures that this link does not exist either. It is therefore created. Both the *p2k* relationship and the new traversal link *newTLink* are passed in the Where clause to the *DefineTraversalLink* rule, which is in charge of completing the definition of the new traversal link, namely specifying the origin and target OO-H navigational classes.

It is important to note that this rule is not deterministic. The reason is that the decision criteria implies that providing the user with navigation through at least 80% of the domain relationships is bound to improve the navigability of the application, regardless of which domain relationships are finally left out of the navigational map. The definition of a deterministic transformation that chose the best candidate relationship to be included in the model (according for example to certain semantic criteria) would require that the navigational meta-model provided such semantic information. Unfortunately, to our knowledge extent, up to now none of the current WE navigation meta-models cover such issues.

The DefineNewTraversalLink Evolution Rule

The last UPT relation, called *DefineNewTraversalLink*, is displayed in Figure 14. This relation involves only the two domains received as parameters: the check-only domain

Association (which reflects the relationship chosen in the previous rule to be the basis of the new traversal link) and the enforceable domain *TraversalLink*, which represents the new link created. The leftmost branch of the relation keeps the name of the *association* parameter in the variable *na*, and the name of the two ends of the relationship in the variables *role1* and *role2*. In addition, it keeps the name of the navigational classes that are supported by the corresponding domain classes in the variables *nav1* and *nav2*. Returning to our example, if we suppose that the previous rule has selected the *p2k* relationship, the value of *na* would be "p2k", the value of *nav1* would be "Publications" and the value of *nav2* would be "Keywords" (see Figure 8). It also has to be remembered that, in a class diagram, in the absence of a role name, the one assumed is that of the class. Therefore, the value of *role1* would be set to "Publication" and the value of *role2* would be set to "Keyword".

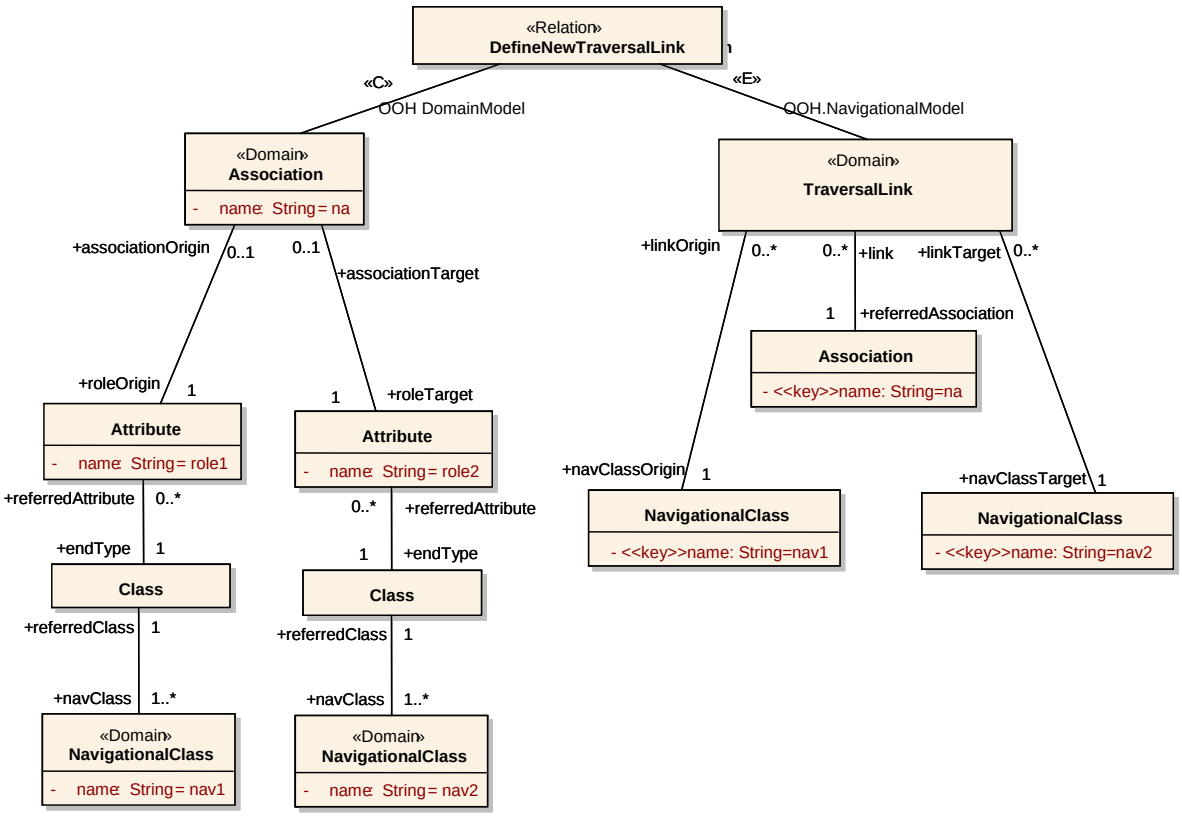

Figure 14: DefineNewTraversalLink Evolution

As we have asserted before, the aim of the enforceable domain in this rule is not to create new objects but just to connect the *newTLink* to the rest of the model elements (in this case the *p2k* relationship whose navigation it must support, as well as the origin and target navigational classes). This is again achieved by means of the <<key>> stereotype. In this rule, it is certain that the rule is going to find the association (in our example the p2k relationship), and therefore just the link between the traversal link and this relationship is created. Similarly, the two *NavigationalClass* instances that correspond to the p2k relationship (Publications and Keywords) are sure to be found, and consequently only the links between those two navigational classes and the new traversal link at meta-model level are established.

Once this final evolution rule has been executed, the control returns to the evaluation rule of Figure 12. The result of assessing the model with the new traversal link 'Publications2Keywords' now gives a DCNM measurement result of 80 (still <=80), which causes the whole sequence to be started again with the aim of finding another association that has not yet been navigated. In this new iteration, there is only one relationship that is not yet supported in the navigational model: the *p2a* relationship. Its inclusion as a new traversal link called 'Publications2Authors' now causes the DCNM measurement result to be 100 (100% of domain relationships are present in the navigational model). This result does not fulfil the When clause of the CheckDCNM rule, and so it finally causes the transformation to stop.

The new navigational model that results from this process can be seen in Figure 15. We now set out to show how all this process is supported by the WebSA Tool that accompanies UPT.

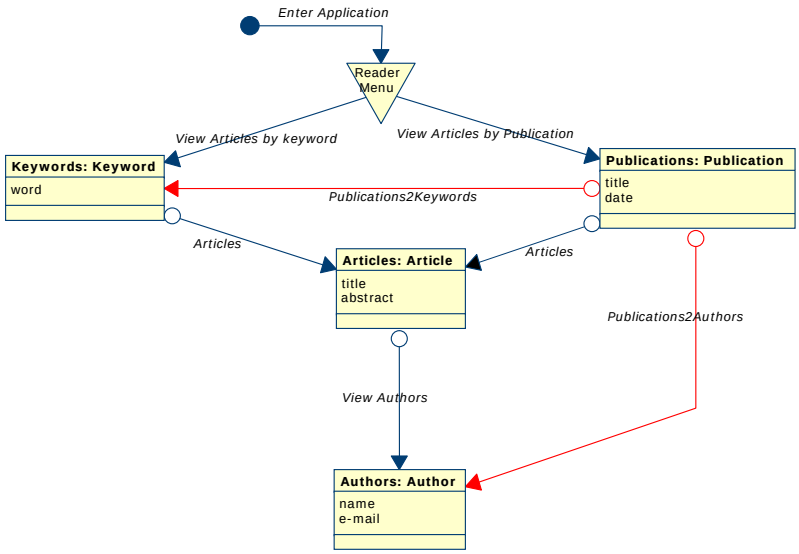

Figure 15: New Navigational Model after the DCNM transformation has been completed

Tool support

We have already made the assertion that an MDE approach must specify the modelling languages, models, translations between models and languages, along with the process used to coordinate the construction and evolution of the models (Kent, 2002). To ensure that the burden of maintaining more than one formal model does not outweigh the benefits of models as tools for abstraction, powerful tool support is required. Additionally, each domain, organization and project may have their own requirements for models, mappings and processes, and therefore the tool should be flexible enough to adapt to this fact. Such is the case of the tool that we have used to provide an implementation of our approach: WebSA Tool (Meliá and Gómez, 2006). WebSA Tool provides support both for *model2text* imperative transformations and *model2model* declarative transformations. Within the context of this paper's aims, only the latter are relevant, however. The process followed in the tool to automate our approach can be seen in Figure 16. The WebSA Tool has pre-built support for the OO-H Domain and

Navigation meta-models, the SMM and the WebSA specific meta-models. Other meta-models for DSML's could be added in the tool with little effort, provided that they are expressible as class diagrams, whose XMI representation is understandable by the tool. The WebSA tool is also capable of generating the code for any transformation expressed in UPT. In order to illustrate this feature, the java code automatically generated by the tool for the transformation DefineNewTraversalLink (see Figure 14) is included in Appendix B. As we can observe in Figure 16, the first step in executing the transformation consists in loading the *Original Navigational Model*, the *Domain Mode*l and the *DCNM Measurement Model* by means of their XMI specification. All the models being stereotyped UML class diagrams, this textual description can be achieved with any UML compliant tool.

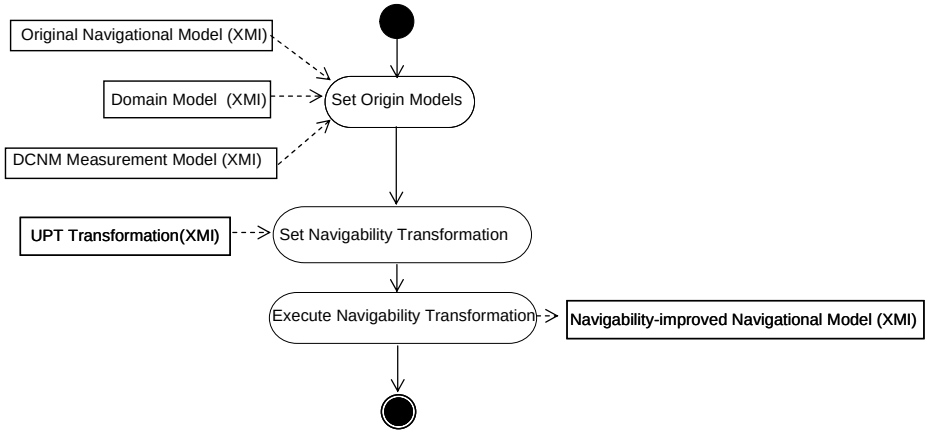

Figure 16: WebSA Tool supporting process

The next step is to provide the transformation itself. Right now the tool only accepts the definition of transformations using UPT. Since this definition is a class model, we can also obtain its XMI specification; this tool must be loaded in the tool. The third step consists in the execution of the transformation. The tool generates the Java code needed to perform the pattern checking/enforcement on the original models, and, if necessary, it also generates the XMI specification of the models that have been changed due to the

transformation execution (in our case just the *Navigability-improved Navigational Model*, see Figure 16). A snapshot of the tool can be seen in Figure 17.

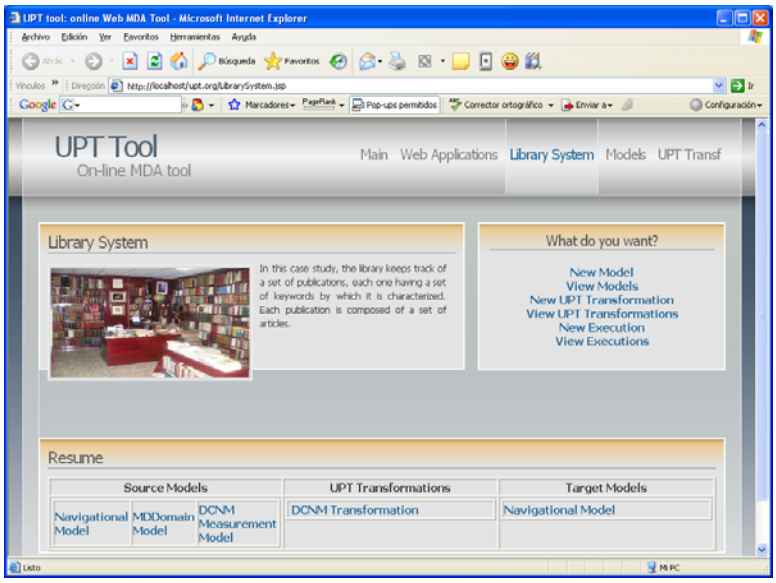

Figure 17: Snapshot of the WebSA Tool

Conclusions and Further Work

In this paper we addressed the question of how to assure the navigability of Web applications from an early stage in development onwards. Navigability in use, understood as the efficiency, effectiveness and satisfaction with which users can move around in the application to satisfy specific goals under specific conditions, is widely recognised as a key attribute for the success of a Web application. The common approach to ensuring navigability is one of applying heuristic evaluation criteria and restructuring rules to the Web application once it has been deployed. This approach comes late, is labor-intensive and is costly.

The engineering approach that we developed is based on two principles: it is measurement-based and it is model-driven. Previous research has proposed navigability measures for navigational models, allowing an early assessment of the navigability of

the Web application that is based on such a model. However, the measures defined were highly dependent on the chosen Web application engineering methodology. A first contribution of our approach is that the method proposed for defining the measures is independent of the Web application engineering methodology used, making our approach more generable and reusable than previous solutions. This independence of particular Web engineering methodologies has been achieved by incorporating into the measure definition process a navigability measurement model that instantiates the Software Measurement Meta-Model (SMM), which is itself derived from the Software Measurement Ontology (SMO). The navigability measures defined are thus stated in the generic terms defined by the SMM and their definition can be translated into the language of a particular Web engineering methodology.

A second contribution of our approach is that it goes further than the mere assessment of a Web application's navigability in an early stage of development. We have shown how navigability measurement models can be integrated in a model-driven Web application engineering process, to assure automatically that the navigational models resulting from that process are compliant with the set of selected decision criteria related to the navigability measures. This process includes a set of model transformations for assessing and bringing about the evolution of navigational models in an automatic way. Including an automatic early navigability assessment and improvement process in Web application engineering methodologies contributes to guaranteeing certain quality levels without significantly increasing development costs or time to market.

To illustrate our approach, the Domain Coverage of the Navigational Model (DCNM) measure as applied to OO-H Navigational Models has been used in this paper. We also showed how this measure can be implemented by a set of UPT model evaluation and

evolution rules that specify a transformation of an OO-H navigational model into a version with an improved DCNM value. We further explained how the execution of these rules can be accomplished by the WebSA tool that provides support for model transformations.

This example, though simple, demonstrates the feasibility of our approach. The main implication of our research for Web application engineering practice is that we showed that ensuring the navigability of Web applications can become an integral aspect of the model-driven development of these applications. We are nevertheless aware of the current limitations of our research and the future work that is needed to implement our approach in Web application engineering practice.

First of all, the proposal of measures is of no value if their practical use is not shown empirically (Basili *et al.*, 1999, Kitchenham *et al.*, 1995, Moody, 2005). Therefore, when tackling the navigability assessment task based on internal measures, we must take into account that the correlation between internal, external and in-use measures is never perfect, and the effect that the value of a given internal navigability measure has upon an associated external/in use navigability measure must be determined empirically (ISO/IEC 9126, 2001). Unfortunately, such empirical research has not been undertaken yet.

Second, the lack of a common ontology and meta-model for Web application engineering makes it necessary to redefine the model transformations for each methodology, which is precisely the aspect that bears the highest workload in our approach. In this sense, we are already working on a common navigation meta-model, as well as on the definition of transformations from this common meta-model to the meta-models defined by particular Web application engineering methodologies, to assure their compatibility.

A third limitation of the approach presented is its focus on syntactic features of Web application models. This limitation was highlighted by the impossibility to define the UPT rule SelectNonNavigatedAssociations (Figure 13), associated to our example DCNM navigability measure, in a deterministic way. The traditional absence of automatic quality assessment in Web application engineering has clearly affected the kind of information that is defined in the different Web engineering meta-models. Semantic aspects that would be taken into account when evolving an application (via its models) to better meet quality criteria, are not always incorporated and thus not available for measurement. The automation of the quality assurance process suggests a need for enrichment of current Web engineering meta-models with semantic information that serves to fine-tune quality-focused model transformations.

Fourth, the WebSA Tool used to illustrate the model-driven aspects of our approach needs to be refined and extended to be more flexible. With that in mind, we are working on WebSA Tool support for other transformation languages apart from UPT, such as QVT. Our approach in this sense envisages providing both the meta-model of the new transformation language and a transformation, defined in UPT, from this meta-model to the UPT meta-model. In this way all the work performed on the execution engine can be reused.

As a final note, we consider our approach to the automatic assessment and improvement of models general, in the sense that it could equally be used with different quality attributes and/or models than navigability. In future research we will develop and evaluate other measures and design evolution patterns that would serve to improve the architectural view of Web applications automatically.

Acknowledgments

Research funding was obtained from the Spanish Ministry of Science and Technology (Grant PR2006-0374 for University Teaching Staff Stages at foreign Universities, projects MEC-FEDER (TIN2004-03145), ESFINGE (TIN2006-15175-C05-05), METASIGN (TIN2004-00779) and DSDM (TIN2005-25866-E)). The research reported upon in this paper is also part of the DADASMECA project (GV05/220), financed by the Valencia Government, and the DADS (PBC-05-012-2) and the DIMENSIONS (PBC-05-012-1) projects, financed by the Regional Science and Technology Ministry of Castilla-La Mancha (Spain).

We would like to thank Silvia Abrahao (Valencia University of Technology) for her insightful comments on the mapping between OO-H and OOWS constructs.

REFERENCES

ABRAHAO S and INSFRAN E (2006) Early Usability Evaluation in Model-Driven Architecture Environments. In Proceedings of the Sixth IEEE International Conference on Quality Software, 287-294, IEEE Press, Wiley, Chichester.

ABRAHAO S, CONDORY-FERNANDEZ N, OLSINA L and PASTOR O (2003) Defining and Validating Metrics for Navigation Models. In Proceedings of the Ninth IEEE International Software Metrics Symposium, 200-210, IEEE Press, Wiley, Chichester.

ASSMANN U, ZSCHALER S and WAGNER G (2006) Ontologies, Meta-Models, and the Model-Driven Paradigm. Ontologies for Software Engineering and Technology. Springer-Verlag, Heidelberg.

AUER T (1999). Quality of IS Use. European Journal of Information Systems, 7(3), 192-201.

BALASUBRAMANIAN K, AGOKHALE A. KARSAI G, SZTIPANOVITS J and NEEMA S. (2006) Developing Applications Using Model-Driven Design Environments. IEEE Computer, 39(2), 33-40.

BASILI V, SHULL F and LANUBILE F. (1999) Building Knowledge through Families of Experiments. IEEE Transactions on Software Engineering, 25(4), 435-437.

BEVAN N. and AZUMA M. (1997) Quality in use: incorporating human factors into the software engineering lifecycle. 3rd International Software Engineering Standards Symposium (ISESS '97). pp. 169-179

BÉZIVIN J (2004) In Search of a Basic Principle for Model-Driven Engineering. Novática 5(2), 21-24.

BRIAND LL, MORASCA S and BASILI V (1999) Defining and Validating Measures for Object-based High-level Design. IEEE Transactions on Software Engineering, 25(5), 722-743.

BRIAND LL, WÜST J, IKONOMOVSKI S and LOUNIS H (1999b) Investigating Quality Factors in Object-Oriented Designs: An Industrial Case-Study. 21st International Conference on Software Engineering, Los Angeles, CA. 345-354

BRIAND LL, WÜST J and LOUNIS H (2001) Replicated case studies for investigating quality factors in object-oriented designs. Empirical Software Engineering, 6 (1) 11-58

BRIAND LL and WÜST J. (2002) Empirical Studies of Quality Models in Object-Oriented Systems. Advances in Computers, Academic Press, 59, 97-166.

CACHERO C (2003). OO-H: Una Extensión a los Métodos OO para el Modelado y Generación Automática de Interfaces Hipermediales. PhD Thesis, University of Alicante, Spain. Available on-line at http://www.dlsi.ua.es/~ccachero/pTesis.htm

CACHERO C (2005). OO-H Navigability Measures. Technical Report, University of Alicante. Available on-line at http://www.dlsi.ua.es/~ccachero/pPublicaciones.htm.

CALERO C, RUIZ J and PIATTINI M (2004) A Web Metrics Survey Using WQM. In Proceedings of the Fourth International Conference on Web Engineering, 147-160, Springer-Verlag, Heidelberg, LNCS 3140.

CAPLAT G and SOURROUILLE JL (2003) Considerations about Model Mapping. In Proceedings of the Workshop in Software Model Engineering, Sixth International Conference on the Unified Modelling Language, Springer-Verlag, Heidelberg, LNCS 2863.

CERI S, FRATERNALI P and BONGIO A (2000) Web Modeling Language (WebML): a Modeling Language for Designing Web Sites. Computer Networks, 33(1-6), 137-157.

COMAI S, MATERA M and MAURINO A (2003) A Model and an XSL Framework for Analyzing the Quality of WebML Conceptual Schemas. In Proceedings of the International Workshop on Conceptual Modeling Quality, 339-350. Springer-Verlag, Heidenberg, LNCS 2784.

EL-EMAM K, BENLARBI S, GOEL N and RAI S (2001) The Confounding Effect of Class Size on the Validity of Object-Oriented Metrics, IEEE Transactions on Software Engineering, 27(7), 630-650.

FENTON NE and PFLEEGER SL (1997) Software Metrics. A Rigurous and Practical Approach. PWS Publications.

FERREIRA M, GARCÍA F, BERTOA M, CALERO C, VALLECILLO A, RUIZ F, PIATTINI M and BRAGA JL (2006). Medición del Software: Ontología y Meta-modelo. Technical Report UCLM-TSI-001, University of Castilla-La Mancha.

FISHER J. (1999) Improving the Usability of Information Systems: the Role of the Technical Communicator. European Journal of Information Systems, 8(4), 294-303.

GARCIA F, BERTOA MF, CALERO C, VALLECILLO A, RUIZ F, PIATTINI M and GENERO M (2005) Towards a Consistent Terminology for Software Measurement. Information and Software Technology, 48(8), 631-644.

GENERO M, MANSO MªE, VISAGGIO A, CANFORA G and PIATTINI M (2007) Building a Metric-based Prediction Model for UML Class Diagram Maintainability, Empirical Software Engineering (to be published).

GOMEZ J, CACHERO C and PASTOR O (2001) Conceptual Modeling of Device-Independent Web Applications. IEEE MultiMedia, 8(2), 26-39.

HENNICKER R and KOCH N (2000) A UML-based Methodology for Hypermedia Design. In Proceedings of the UML 2000, 410-424, Springer-Verlag, Heidelberg, LNCS 1939.

HEUSER L (2004) The Real World or Web Engineering? In Proceedings of the Fourth International Conference on Web Engineering, 1-5, Springer-Verlag, Heidelberg, LNCS 3140.

ISO/IEC 9126 (2001) Software Engineering – Product quality – Part 1: Quality Model. International Organization for Standardization, Geneva.

IVORY MY (2004) Automated Web Site Evaluation. Kluwer Academic Publishers, Norwell.

IVORY MY and HEARST MA (2001) The State of the Art in Automating Usability Evaluation in User Interfaces. ACM Computing Surveys, 33(44), 470-516

IVORY MY, MEGRAW R (2005) Evolution of Web Site Design Patterns. ACM Transactions on Information Systems, 23(4), 463-497.

KENT S (2002) Model-Driven Engineering. In Proceedings of the Third International Conference on Integrated Formal Methods, p 286-298, Springer-Verlag, Heidelberg, LNCS 2335.

KITCHENHAM B, PFLEEGER S and FENTON N (1995) Towards a Framework for Software Measurement Validation. IEEE Transactions of Software Engineering, 21(12), 929-943.

KOCH N and KRAUS A (2003) Towards a Common Meta-model for the Development of Web Applications. In Proceedings of the Third International Conference on Web Engineering, 497-506, Springer-Verlag, Heidelberg, LNCS 2722.

KOCH N (2006) Transformation Techniques in the Model-Driven Development Process of UWE. In Proceedings of the Sixth International Conference on Web Engineering, ACM Press, New York.

LANG M and FITZGERALD B (2005) Hypermedia Systems Development Practices: A Survey. IEEE Software, 22(2), 68-75.

MANSO MªE, GENERO M. and PIATTINI M. (2003) No-Redundant Metrics for UML Class Diagrams Structural Complexity. In Proceedings of the CAISE 2003, 127-142, Springer-Verlag, Heidelberg, LNCS 2681.

MELIÁ S, GÓMEZ J and SERRANO JL. (2006) UPT: A Graphical Transformation Language based on a UML Profile. In Proceedings of the Second European Workshop on Milestones, Models and Mappings for Model-Driven Architecture, 81-97, Springer-Verlag, Heidelberg.

MELIA S, KRAUS A and KOCH N (2005) MDA Transformations Applied to Web Application Development. In Proceedings of the Fifth International Conference on Web Engineering, ACM Press, New York.

MELLOR SJ, SCOTT K, UHL A and WEISE D (2004) MDA Distilled. Principles of Model-Driven Architecture. Addison-Wesley Professional, Boston.

MOODY DL (2005) Theoretical and Practical Issues in Evaluating the Quality of Conceptual Models: Current State and Future Directions. Data and Knowledge Engineering, 55(3), 243-276.

MYERS BA and ROSSON MB (1992) Survey on User Interface Programming. In Striking a Balance. In Proceedings of the International Conference for Human-Computer Interaction (CHI'92), 195-202, ACM Press, New York.

Nielsen J (2000) Designing Web Usability: The Practice of Simplicity. New Riders, Berkeley.

OMG-OCL (2006) Object Constraint Language (OCL). OMG doc. formal/2006-05-01.

OMG-MOF (2006) Meta Object Facility (MOF) v2.0. OMG doc. formal/2006-01-01.

OMG-QVT (2005) MOF QVT Final Adopted Specification. OMG doc. ptc/05-11-01

OMG-UML (2005) UML 2.0 Superstructure Specification. OMG doc. formal/2005-07-04.

OMG-XML (2005) XML Metadata Interchange (XMI) Specification v2.1. OMG doc. formal/2005-09-01.

PASTOR O, ABRAHAO S and FONS JJ (2001) An Object-Oriented Approach to Automate Web Applications Development. In Proceedings of the Second International Conference on Electronic Commerce and Web Technologies, 6-28, Springer-Verlag, Heidelberg, LNCS 2115.

PLANET MDE (2005) MDE (Model Driven Engineering). Community Portal. Available electronically at: http://www.planetmde.org (last accessed in May 2007).

POELS G and DEDENE G. (2001) Evaluating the Effect of Inheritance on the Modifiability of Object-Oriented Business Domain Models, In Proceedings of the Fifth European Conference on Software Maintenance and Reengineering (CSMR 2001), Lisbon, Portugal, 20-28.

SCHAUERHUBER A, WIMMER M and KAPSAMMER E (2006) Bridging Existing Web Modeling Languages to Model-Driven Engineering: A Meta-model for WebML. In Proceedings of the Second Model-Driven Web Engineering Workshop, ACM Press, New York.

SCHMIDT DC (2006) Model-Driven Engineering. IEEE Computer, 39(2), 25-31.

SINGH SN, DALAL N and SPEARS N (2005) Understanding Web Home Page Perception. European Journal of Information Systems, 14, 288–302.

Appendix A: QVT-Relation specification of UPT Transformations

In Table 5 we are presenting the QVT-Relation specification that is automatically generated from the UPT specification of the three rules that implement the DCNM measure in the context of OO-H (see Figure 12, Figure 13 and Figure 14).

```
transformation DCNMTransformation (smm:SMM, ohhd:OOHDomain, oohn: OOHNavigation)
{
  key TraversalLink {name};
  key NavigationalClass {name};
  key Association {name};
  key Class {name};

  top relation CheckDCNM
  {
   checkonly domain i:Indicator {
            description='Nav_L',
            dcnm = drm:DerivedMeasure{
       description='DCNM',
             analisysM= am:AnalysisModel {
                        description='AM_NavL',
                        decision= dc:DecisionCriteria {
                           description = 'DCNM<=80->DCNM_L=Not Acceptable
                                          and DCNM>80->DCNM_L=Acceptable'
                        }
             },
       mmethod = mm1:MeasurementMethod {
                        description = "(NNDR/NRR) *100',
                        bm = bm1:BaseMeasure { description = 'NNR',
                          mmethod = mm2:MeasurementMethod {description='CountNNR'}
                        },
                        bm = bm2:BaseMeasure { description = 'NNDR',
                         mmethod = mm3:MeasurementMethod {description='CountNNDR'}
                        }

            }
     }
  };
  checkonly domain dm:DomainModel {
            associations = aso1:Association {
                        roleOrigin = a1:Attribute {
                                    endType = Class {coveredClass = nc1:NavigationClass{}}
                        },
                        roleTarget = a2:Attribute {
                                    endType = Class {coveredClass = nc2:NavigationClass{}}
                        }
  };
  checkonly domain nm1:NavigationalModel{
                        tlinks = tl:TraversalLink{
                                    converedAssociations = aso2:Association {}
                        }

  };
  when {
            nm1.tLinks.coveredAssociations->asSet()->size()/dm.associations->size()*100<=80
  }
  where {
            CheckNonNavigatedAssociations (aso1, tl,nm1);CheckDCNM(i,dm, nm1)

  }
 }

  relation SelectNonNavigatedAssociations {
```

```
            n1, n2, nA, nB: Class;
            nav1, nav2: String;
            checkonly domain aso: Association {
                        roleOrigin = a1:Attribute { endType = c1:Class{name = n1}},
                        roleTarget = a2:Attribute { endType = c2:Class{name = n2}}
      };
            checkonly domain tl: TraversalLink {
                        navClassOrigin = nc1:NavigationalClass {
                                    name = nav1,
                                    referredClass = rc1:Class {name = nA}
                        },
                        navClassTarget = nc2:NavigationalClass {
                                    name = nav2,
                                    referredClass = rc2:Class {name = nB}
                        }

            };
            enforceable domain nm1: NavigationalModel {
                        tlinks = newTLink:TraversalLink { name = nav1 + "2" + nav2 }
            };
            when {
                        Not((nA=n1 and nB=n2) or (nA=n2 and nB=n1))
            }
            where {
                        DefineNewTraversalLink(aso,newTLink)
            }
  }



  relation DefineNewTraversalLink {
            na, nav1, nav2, role1, role2: String
            checkonly domain aso: Association {
                        name = na,
                        roleOrigin = a1:Attribute {
                                    name = role1,
                                    endType = Class {
                                                navClass = nco1:NavigationalClass {name = nav1}
                                    }
                        },
                        roleTarget = a2:Attribute {
                                    name = role2,
                                    endType = Class {
                                                navClass = nco2:NavigationalClass {name = nav2}
                                    }
                        }
            };
            enforceable domain tl: TraversalLink {
                        navClassOrigin = nct1:NavigationalClass {name = nav1},
                        navClassTarget = nct2:NavigationalClass {name = nav2},
                        referredAssociation = nct3:NavigationalClass {name = na}
            };
  }


}
}
```

Table 5 QVT-Relation Specification of CheckDCNM Rule

As an example, the QVT-Relation corresponding to Figure 12 is translated into a *top relation* block in Table 5. Inside this block, each branch of the relation is preceded by their nature (*checkonly, enforceable*). Inside each branch, we find a set of either simple pairs attribute-value or complex structures that are further decomposed until reaching

the attribute-value level of detail. Last, *when* and *where* clauses are additional blocks of the QVT-Relation specification.

Appendix B: Java Code corresponding to the DefineNewTraversalLink Transformation

In order to illustrate the operationalization of the UPT transformations in the WebSA tool, in Table 6 we present the Java Code, automatically generated by the WebSA Tool, corresponding to the DefineNewTraversalLink Transformation.

```
Package utilidades.resultados;

import ooh.*;
import ooh.conceptualviewooh.*;
import ooh.navigationalviewooh.*;

public class DefineNewNavigationalLink{

  VariablesDomain variablesDomainCheckable = new VariablesDomain();
  Propiedad propiedadAux = null;

  public boolean execute(Association association, NavigationalAssociation navigationalAssociation, OOHPackage oOH){
    boolean result = true;

    // The checkable OOH.DomainView generation starts
    TableVariables variablesObjectCheckableAssociation = new TableVariables(association, "association", "Association");
    boolean saveTableObjectCheckableAssociation = true;
    propiedadAux = new Propiedad("na", association.refGetValue("name"));
    variablesObjectCheckableAssociation.addPropiedad(propiedadAux);
    if (saveTableObjectCheckableAssociation){
      saveTableObjectCheckableAssociation = false;
      Attribute roleTarget = (Attribute) association.getRoleTarget(); //It navigates through the roleTarget
      if (roleTarget != null){
        TableVariables variablesObjectCheckableRoleTarget = new TableVariables (roleTarget, "roleTarget", "Attribute");
        boolean saveTableObjectCheckableRoleTarget = true;
        propiedadAux = new Propiedad("role2", roleTarget.refGetValue("name"));
        variablesObjectCheckableRoleTarget.addPropiedad(propiedadAux);
        if (saveTableObjectCheckableRoleTarget){
          saveTableObjectCheckableRoleTarget = false;
          Class endType = (Class) roleTarget.getEndType();
          if (endType != null){
            TableVariables variablesObjectCheckableEndType = new TableVariables (endType, "endType", "Class");
            boolean saveTableObjectCheckableEndType = true;
            if (saveTableObjectCheckableEndType){
              saveTableObjectCheckableEndType = false;
              for (java.util.Iterator i2 = endType.getNavClass().iterator(); i2.hasNext(); ){
                NavigationalClass navClass = (NavigationalClass) i2.next();
                TableVariables variablesObjectCheckableNavClass =
  new TableVariables(navClass, "navClass", "NavigationalClass");
                boolean saveTableObjectCheckableNavClass = true;
                propiedadAux = new Propiedad("nav2", navClass.refGetValue("name"));
                variablesObjectCheckableNavClass.addPropiedad(propiedadAux);
                if (saveTableObjectCheckableNavClass){
              variablesObjectCheckableEndType.addRelacion(variablesObjectCheckableNavClass);
                    saveTableObjectCheckableEndType = true;
                }
              }
            }
            if (saveTableObjectCheckableEndType){
              variablesObjectCheckableRoleTarget.addRelacion(variablesObjectCheckableEndType);
```

```
            saveTableObjectCheckableRoleTarget = true;
           }
          }
         }
         if (saveTableObjectCheckableRoleTarget){
          variablesObjectCheckableAssociation.addRelacion(variablesObjectCheckableRoleTarget);
          saveTableObjectCheckableAssociation = true;
         }
        }
       }
       if (saveTableObjectCheckableAssociation){
        saveTableObjectCheckableAssociation = false;
        Attribute roleOrigin = (Attribute) association.getRoleOrigin();
        if (roleOrigin != null){
         TableVariables variablesObjectCheckableRoleOrigin = new TableVariables(roleOrigin, "roleOrigin", "Attribute");
         boolean saveTableObjectCheckableRoleOrigin = true;
         propiedadAux = new Propiedad("role1", roleOrigin.refGetValue("name"));
         variablesObjectCheckableRoleOrigin.addPropiedad(propiedadAux);
         if (saveTableObjectCheckableRoleOrigin){
          saveTableObjectCheckableRoleOrigin = false;
          Class endType = (Class) roleOrigin.getEndType();
          if (endType != null){
           TableVariables variablesObjectCheckableEndType = new TableVariables(endType, "endType", "Class");
           boolean saveTableObjectCheckableEndType = true;
           if (saveTableObjectCheckableEndType){
            saveTableObjectCheckableEndType = false;
            for (java.util.Iterator i3 = endType.getNavClass().iterator(); i3.hasNext(); ){
             NavigationalClass navClass = (NavigationalClass) i3.next();
             TableVariables variablesObjectCheckableNavClass =
    new TableVariables(navClass, "navClass", "NavigationalClass");
              boolean saveTableObjectCheckableNavClass = true;
              propiedadAux = new Propiedad("nav1", navClass.refGetValue("name"));
              variablesObjectCheckableNavClass.addPropiedad(propiedadAux);
              if (saveTableObjectCheckableNavClass){
             variablesObjectCheckableEndType.addRelacion(variablesObjectCheckableNavClass);
               saveTableObjectCheckableEndType = true;
              }
             }
            }
            if (saveTableObjectCheckableEndType){
             variablesObjectCheckableRoleOrigin.addRelacion(variablesObjectCheckableEndType);
             saveTableObjectCheckableRoleOrigin = true;
            }
           }
          }
          if (saveTableObjectCheckableRoleOrigin){
           variablesObjectCheckableAssociation.addRelacion(variablesObjectCheckableRoleOrigin);
           saveTableObjectCheckableAssociation = true;
          }
         }
        }
        if (saveTableObjectCheckableAssociation){
         variablesDomainCheckable.addTable(variablesObjectCheckableAssociation);
        } else result = false;
        // The checkable OOH.DomainView generation ends

        if (variablesDomainCheckable.getTables().isEmpty())  result = false;


        //The enforceable OOH.NavigationalView generation starts
      navigationalAssociation.refSetValue("name", variablesDomainCheckable.obtenerValorVariable("role1") + "2" +
     variablesDomainCheckable.obtenerValorVariable("role2"));

        Association referredAssociation = null;
        for (java.util.Iterator it0 = referredAssociation.refAllOfClass().iterator(); it0.hasNext(); ){
         Association referredAssociationAux = (Association) it0.next();
         try {
          if (referredAssociationAux.getName().equals(variablesDomainCheckable.obtenerValorVariable("na"))){
           referredAssociation = referredAssociationAux;
           break;
          }
         } catch(NullPointerException e){
          System.out.println ("The association with the name na has not been found");
    }
        }
        if (referredAssociation == null){
```

```
      referredAssociation = referredAssociation.createAssociation();
      referredAssociation.refSetValue("name", variablesDomainCheckable.obtenerValorVariable("na"));
    }

    navigationalAssociation.setReferredAssociation(referredAssociation);
    NavigationalClass navClassOrigin = navigationalAssociation.createNavigationalClass();
    navClassOrigin.refSetValue("name", variablesDomainCheckable.obtenerValorVariable("nav1"));
    navigationalAssociation.setNavClassOrigin(navClassOrigin);
    NavigationalClass navClassTarget = navigationalAssociation.createNavigationalClass();
    navClassTarget.refSetValue("name", variablesDomainCheckable.obtenerValorVariable("nav2"));
    navigationalAssociation.setNavClassTarget(navClassTarget);

//The enforceable OOH.NavigationalView generation ends

    return result;
  }


}
```

Table 6 Java Code corresponding to the DefineNewTraversalLink Transformation