ISBN 978-83-60810-51-4

A Feature Model Configuration for Multimedia Applications by an OWL-based Approach

Giuseppe Santoro*, Carmelo Pino*, Concetto Spampinato *
*Department of Electrical, Electronics and Informatics Engineering University of Catania,
Viale Andrea Doria, 6 - 95125 Catania, Italy

***Abstract*—Feature models are used to describe common and variable properties of families of related software systems referred as SPL (Software Product Line). Every program in an SPL is identified by a unique and legal combination of features called feature configuration. There is no formal semantic for describing a feature model and no standard tool for building and validate a feature configuration. In this paper we present an OWL-based approach for building and editing feature models together with an OWL-based inferential engine for creating a feature configuration and check its consistency. The Framework has been developed as a Java web service and can be applied to model a wide range of applications from media processing to business and financial modeling.**

I. Introduction

THERE is no agreed upon a standard semantic for feature models and that makes difficult to exchange and share feature models in practical applications [1]. Many variations to the original feature model notation FODA (Feature Oriented Reuse Method) [2] have been proposed as FORM (Feature Oriented Reuse Method) and FeatuRSEB [1]. Lack of standard semantic is the principal cause of the absence of a standard automated tool for checking the correctness of a feature configuration. In [3] the authors propose an approach to propagate constraints and provide automatic selection and justification for automatically selected/deselected features through Logic based Truth Maintenance System (LTMS) and Boolean Satisfiability Problem Solver (SAT solver). In [4] feature models are transformed into a Constraint Satisfaction Problem and a constraint solver is used to obtain a valid feature configuration. A few ontology based approaches [5], [6] has been found in literature and they will be discussed in detail in section III. In this paper we present an OWL-based tool capable of inferring a valid feature configuration from some user defined features. We describe also a feature model semantic that helps to implement this engine and some tools to support feature model building and diagram visualisation. This paper is organised as follows. Section II introduces feature model. Section III illustrates the OWL ontology used for representing feature models. In section IV we describe the entire framework with a particular attention to the tools that enable building and editing a feature model and creating an SVG graphical representation of a feature configuration. Section V describes the inferential engine for building a valid configuration from some user defined features. Section VI gives some technical details about the implementation choices. Concluding remarks are given in section VII.

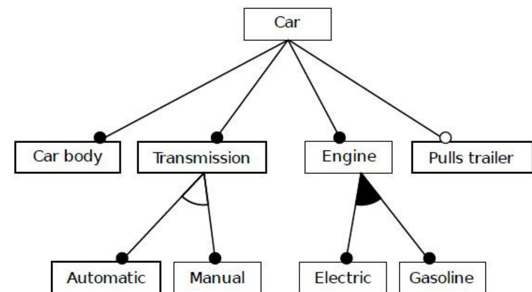

Fig. 1. Feature Model example taken from [7]

II. Feature Models

A feature is defined as an important property of a concept [7] where by concepts, we mean anything in the domain of interest. In the description of a software system, features may occur at any level, e.g. high-level system requirements, architectural level, subsystem and component level, implementation-construct level (e.g. object or procedure level) [7]. Feature modelling is the activity of modelling the common and the variable properties of concepts and their interdependencies and organising them into a coherent model referred as a feature model. [7]. A feature model consists of a tree diagram called feature diagram with some additional textual information, such as semantic description. A feature diagram is made up of nodes, directed edges and edge decorations, where the root node represents the subject we want to describe, formally called concept, and the remaining nodes are its features. Features are further connected by edges to subfeatures in an hierarchical structure. Formally an instance of a feature diagram is known as a concept description or a feature configuration and is defined valid if it does not break feature constraints and inclusion rules.

A. Feature diagram example

The example we are going to discuss through the entire paper has been taken from [7] and describes commonalities and differences among instances of a Car (fig.1).

In order to represent this example, we have chosen to use Czarnecki's notation with decorated edges, because it is the best know formalism in literature. The concept *Car* is described by features Car *Body*, *Transmission*, *Engine* and *Pulls trailer*. A *Transmission* feature is further described by

subfeatures *Automatic* and *Manual*; an *Engine* is described by subfeatures *Electric* and *Gasoline*. Without a way to add some restrictions on the feature model this diagram also describes invalid cars like one with a transmission that is both automatic and manual. Semantic restrictions like those are provided by different edge decorations (see Section II-B). A valid feature configuration of this example can be described by the features: *Car body*, *Transmission*, *Automatic*, *Engine*, *Electric*. Such a configuration do not violate model restrictions and so a car with a car body, an automatic transmission and an electric engine can be manufactured. Besides this one, such diagram allows cars that pulls a trailer and hybrids cars, i.e cars with an engine that is both electric and gasoline.

B. Feature diagram entities

Many variations to the original feature model notation FODA (Feature Oriented Reuse Method) [2] have been proposed, as FORM (Feature Oriented Reuse Method) and FeatuRSEB [1], but none of them has been accepted as a standard. In this project we have chosen to use Czarneckis notation without edges decoration as a starting point and we introduced some new feature constraints (see Section II-C). We give here a brief description of each feature type and their selection rules with Czarneckis notation with decorated edges. Like in section II-A, we have chosen this notation and not that one of our project because of its widespread coverage. Besides with this formalism we have also the chance to explain why we preferred this one over the others. As in FODA, Czarnecki distinguishes between *mandatory*, *alternative* and *optional* features, but he introduces also *Or* features. A *Mandatory* Feature (see Fig.1 at features *Engine* or *Car body*) is included in a feature configuration if and only if its parent is included as well. It is represented graphically by a simple edge without decorations ending with a filled circle. An *Optional* Feature (see Fig.1 at feature *Pulls Engine*) may be included in a feature configuration if and only if its parent is included. It is represented by a simple edge without decorations ending with an empty circle. Only one feature in a set of Alternative Features (see Fig. 1 at features *Automatic* and *Manual*) can be included in a configuration. *Alternative* Features are represented by edges connected by an arc. In a set of *Or* Features (see Fig. 1 at features *Electric* and *Gasoline*) any non-empty subset of features can be included in a configuration. Or Features are represented by edges connected by a filled arc. Besides these features, we have also *Optional Alternative* Features, when there is at least one *optional* Feature in a set of *Alternative* Features (see the left side of normalisation at Fig.2), and *Optional Or* Features, when there is at least one *optional* feature in a set of *Or* Features (see the left side of normalisation at Fig.3).

A feature diagram with one or more *Optional Alternative* Feature is normalised into a diagram with all *Optional Alternative* features (see Fig.2). A feature diagram with one or more *Optional Or* features is normalised into a diagram with all *Optional Or* features which is equivalent to have all features optional (see Fig.3). So the category of *Optional Or* features

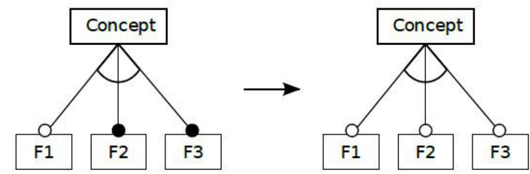

Fig. 2. Optional Alternative Features normalisation

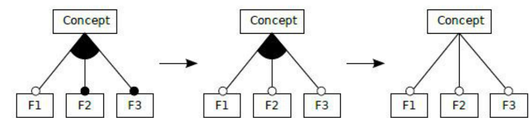

Fig. 3. Optional Or Features normalisation

equivalent to the category of *Optional* features.

C. Feature diagram notation

In this section we introduce our new formalism with a representation of the running example (see Fig.4).

This notation comes from Czarneckis [7] feature diagram notation without edge decorations but contains also some three new features constrains. Every feature model created with the previous formalism can be converted to an equivalent feature diagram without edge decorations. We used this formalism, even if it is less concise than the other one, because of its simpler structure and a simpler analysis required. The simpler structure is due to type of information not stored in a feature itself but in its parent node, so that every node has an homogeneous set of subfeatures. As consequence of this structure, if you start using it from the beginning, no normalisation is required. In this notation concepts, parent nodes of mandatory features and leaf features (features without subfeatures) are represented like a filled circle (see Fig. 4 at features Car, Car And and Body); parent nodes of optional features with an empty circle (see Fig. 4 at feature Car Opt); parent nodes

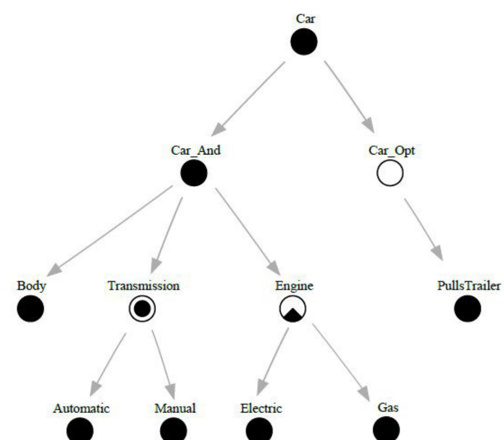

Fig. 4. Car example without edge decorations

of alternative features with two concentric circle, where the internal one is filled (see Fig. 4 at feature *Transmission*); parent nodes of optional alternative features are represented like alternative features but with the internal circle empty; parent nodes of or-features are represented with a more complex figure (see Fig. 4 at feature Engine). In this new diagram there are two new nodes *Car* And and *Car Opt* that were not there in the previous one. These nodes do not represent an entity of the world but are only used for grouping features under a common feature type. This kind of node is called *FeatureGroup* and like every other node with child features (also called node feature) have a particular representation depending from the type of subfeatures. Selection rules with this notation are equivalent to those introduced in section II-B, but now rules on a parent node influences selection of its children. Besides a concept node and all its direct features are always selected in a feature configuration. Feature type rules are not the only restrictions that influences feature configuration construction. Constraints can exists between features in different branches of a diagram tree. Czarnecki [7] enriched the original FODA notation with two kind of feature constraints: *mutual-exclusion* constraints and *requires* constraints, renamed here respectively *excludes* constraints and *implies* constraints. Those two types of constraints are modified to be unidirectional but maintains same semantic. We have also introduced in our formalism two other kind of constraints *avoid* and *default*. Implies and excludes constraints are binary and unidirectional ones whereas avoid and default are unary. In a feature model *A implies B* means that the existence of a feature A in a feature configuration implies the existence of a feature B; whereas A excludes B means that if a feature A is included in the configuration feature B should be not included; Avoid A means that the feature A should not be in the feature configuration and default A means that the feature A should be in a configuration by default.

III. FEATURE MODELS IN OWL

In this section we want to illustrate the OWL-based approach we have developed to represent and manage feature models. OWL stand for Web Ontology Language and is the standard defacto for the semantic web. Its expressive power and formal semantics make it usable in many other domains [5]. It consists of three increasingly expressive sub-languages: OWL Lite, DL and Full. We use the OWL DL dialect because we want to infer a valid feature configuration using a DL reasoner (see Section V). In the literature we found two main approaches for representing a feature model through an OWL ontology and checking its consistency. The first approach [6] represent features in a diagram like an OWL class and every feature relations like an object properties. For example in order to represent a car with a transmission (see Section II-A and Fig. 1) you should create two OWL classes *Car* and *Transmission* and an object property *hasTransmission* for representing this relation. This approach does not represent features in a configuration as instances of classes (OWL individual) as intuitively you would think. OWL classes are

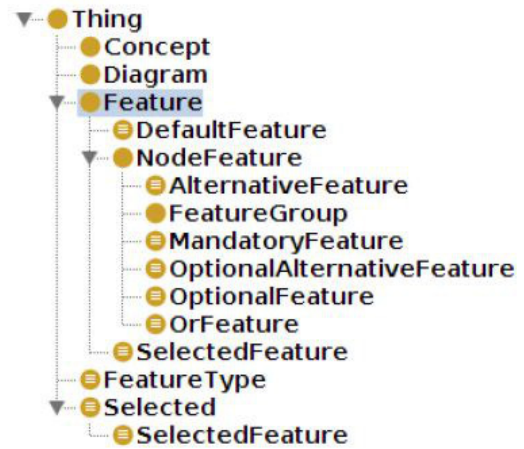

Fig. 5. Feature model ontology classes

used to simulate features in order to use TBox (terminological box or class-level) reasoning for checking consistency. This solution was justified by limitations of the OWL reasoner RACER [1] in the 2005. At that time RACER was only able to detect ABox (assertional box or instance level) inconsistencies but not which classes has caused them. The second approach [5] represents feature models in a descriptive way, mapping every feature in a model or in a configuration with an instance of class Feature or one of its subclasses. This solution was proposed in the 2008 with another reasoner called Pellet [8], [2]. This reasoner was already able of both ABox and TBox reasoning and debugging, overcoming the limitation of the previous approach. We have chosen to use the latter approach because it simplifies simplify the representation of feature hierarchies and more important because it makes possible to express SWRL consistency rules on OWL classes (see Section V) and infer, a valid feature configuration through SWRL rules.

A. Feature model ontology

Every feature model or configuration is an instance of a schema defined in an OWL file. Such schema describes vocabulary, structure and type restrictions of feature models (see Fig. reffig5), i.e. features cannot be added as children of a diagram node.

The main ontology class of this schema is Feature, which represents all kind of nodes in a feature diagram except for the root node which is a concept. Because of the Open World Assumption [11] only node features are described in our ontology (see Fig. 5 at the NodeFeature class). Every feature that is not a node feature is considered as leaf feature. Our formalism without edge decoration enables to represent edges like a normal parent/child relation without any type of information through object properties *hasChild* and *hasParent*. In order to ensure that a feature diagram is a tree and not

[1]http://www.sts.tu-harburg.de/ r.f.moeller/racer/
[2]http://clarkparsia. com/pellet/

a graph we introduced some restrictions on these properties: *hasParent* is defined as a functional property, meaning that an individual of its domain can only have one parent; *hasChild* is defined an inverse functional property, specifying that two different parents cannot share the same child node; both these properties are declared irreflexive, which enables to avoid an individual have itself as a child or a parent. Feature type information are stored on diagram node with the object property *hasFeatureType*. It is defined functional because it can connect a Feature instance with only one of the individuals of class FeatureType (see Fig. 5 at the FeatureType class): *Mandatory*, *Optional*, *Alternative*, *Or* and *OptionalAlternative*. The main subclass of Feature is *NodeFeature*, which represents a feature with at least one subfeature. A *NodeFeature* instance can be further classified according to feature types in: *MandatoryFeature*, *OptionalFeature*, *AlternativeFeature*, *OrFeature* and *OptionalAlternativeFeature*. This new formalism introduces a new feature diagram entity called *FeatureGroup* (see Fig. 5 at the *FeatureGroup* class), which does not represent a real diagram entity but only a way to group features under a common feature type. A feature group inhereditates from the super class NodeFeature and must have at least a child feature. Features can be classified in *SelectedFeature* and *DefaultFeature*, respectively, according to the properties isSelected and default. To the first group belong features that have been selected by the user to be in a feature configuration. To the second group belong features that the user want to be selected automatically by the inferential engine during the feature diagram construction. Besides feature class, we introduce two other classes: Concept and Diagram. A concept must contain at least a child node of type *FeatureGroup*, while a diagram must contain at least a concept. In order to support feature constraints (see Section II-B), we have introduced the following properties: *implies*, *excludes*, *default* and *avoid*. Implies and excludes are object properties and they connect two instances of the Feature class. They are declared irreflexive, so that a feature cannot implies or excludes itself. *Avoid* and *default* are boolean data properties and they are applied only to feature instances. Both these properties are declared *functional*, so that they cannot be declared twice on the same feature instance. Avoid property is used to define which feature we want to deselect from a feature model. Default properties are used to define a feature that will be automatically selected by the inferential engine. This property is very useful for *Or*, *alternatives* and *optional alternative* feature in order to select automatically one of the subfeatures. We provide also two other object properties: next and previous, which enable sorting irreflexive features in a diagram.

IV. Feature Model Ontology Framework

Our feature model framework is made of five modules: *ModelManager*, *ModelBuilder*, *InconsistencyChecker*, *SelectionEngine* and *SVGDiagramBuilder*. ModelManager is the main module of the entire application and it is used by other module in order to load and save local or remote feature models. This module allows also to obtain an OWL DL reasoner used for building the inferential engine (see Section V) or to serialise a feature model to a string. *ModelBuilder* is used for building a new feature ontology model or editing an existing one. Some common allowed operations are:

- creating/deleting diagram nodes, i.e. diagram, concept, leaf node, feature group.
- adding/removing a parent/child relation between two nodes, i.e. add a concept to a diagram or remove a leaf feature to a node feature.
- setting/changing the feature type of a feature node.
- adding/removing feature constraints between two features, i.e. implies, excludes, avoid or default constraints.
- selecting/deselecting a feature in a feature configuration, i.e. set the data property isSelected to true/false.

Every operation is followed by an inconsistency check, in order to detect model or OWL conflicts, i.e. adding to a diagram a feature instead of a concept or adding an irreflexive feature constraint between a feature and itself. Such conflicts are induced by OWL restrictions in the feature model schema (see Section III-A). If a conflict is detected, an exception is launched with the OWL individual, which cause the inconsistency, and the SWRL rule violated. Such exceptions are raised also for additional user defined inconsistency (see InconsistencyChecker) and selection rules inconsistency (see Section V). The InconsistencyChecker module is used to check additional SWRL inconsistencies rules on a feature model. SWRL rules files can be loaded/removed dynamically at runtime and inconsistency is checked through a OWL DL reasoner. SVGDiagramBuilder and SelectionEngine modules are described in detail respectively, in sections IV-A and V.

A. SVGDiagramBuilder

This module is optional and its used only to get a graphic representation of a feature diagram. It takes as input the URI of a feature diagram within an OWL ontology file and produces an SVG vectorial image. Building such graphic representation requires three steps: loading the feature model which the diagram belongs to; building with the language Graphviz/DOT [3] a textual representation of the diagram (see Fig.6); compiling the DOT representation into an SVG image (see Fig.7 for the textual representation and Fig.4 for the graphical representation).

The first step is performed by the module ModelManager and it requires the URI of the OWL file containing the feature diagram. Building a DOT representation of the diagram implies mapping each node in feature diagram into a node with a label, representing the name of the feature, and an image, representing the type of node (see Section II-C). In a feature configuration, two different colours, i.e. grey and black, represent deselected/selected features or concepts. The last step is performed with the command dot-Tsvg. You can use this command specifying an input DOT file and an output SVG file, but we prefer not to create intermediate DOT files. Thus we use this command write our DOT diagram to the

[3]http://www.graphviz.org/

```
digraph CarModel {
   ranksep=0.6; nodesep=0.5; bgcolor=white;
   colorscheme=svg;
   fontsize=17; label="CarModel"; labelloc=t;

   node [regular=true, width=0.8, fixedsize=true,
      style=invis];
   edge [fontsize=10, labelangle=-135,
      labeldistance=1.2, color=gray68];

   Car [image="Concept.svg"];
   Car:n -> Car:n [taillabel="Car", color=transparent,
      constraint=false]

   Car_Opt [image="Optional.svg"];
   Car_Opt:n -> Car_Opt:n [taillabel="Car_Opt",
      color=transparent, constraint=false]
   ...

   Car -> Car_Opt;
   Car -> Car_And;
   Car_And -> Transmission;
   ...

   { rank = same; Car_Opt; Car_And;  }
   { rank = same; Gas; Electric;  }
   ...
}
```

Fig. 6. DOT representation of a feature diagram

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink">
<g id="graph0" class="graph" ...>
<title ...>CarModel</title>
<text ...>CarModel</text>
<!-- Car -->
<g id="node1" class="node"><title ...>Car</title>
<use xlink:href="#Concept.svg" .../>
</g>
...
<!-- Car&#45;&gt;Car_Opt -->
<g id="edge24" class="edge">
<title ...>Car-&gt;Car_Opt</title><path .../>
<polygon style="fill:#adadad;stroke:#adadad;" .../>
</g>
...
</g>
</svg>
```

Fig. 7. SVG representation of a feature diagram

standard input and getting the SVG representation from the standard output. SVG images are vectorial images serialised in an XML dialect.

V. Selection Engine

This is the main module of the application and it consist of three parts: a general algorithm, an OWL DL reasoner, which you can get with the ModelManager module, and some SWRL rules. The SelectionEngine has the objective of creating automatically a valid feature configuration from some users defined features. In order to be defined valid, such feature configuration should not violates SWRL selection rules (see Section II-B) and OWL model consistency (see Section II-B). A feature configuration is created selecting/deselecting some features in a feature model, thus model consistency is verified during the feature model construction (see Section IV at ModelBuilder).

A. *Selection Algorithm*

The algorithm used by the *SelectionEngine* requires as input an inconsistency free feature model optionally with some feature constraints. Infact every model inconsistency do not allow the OWL reasoner to be used. This algorithm consist of the following steps:

1) The user requires to select/deselect a feature or a concept from the feature model.
2) The SelectionEngine selects/deselects the required node according to the SWRL rules, verifying that it does not contain inconsistencies caused by selection rules or feature constraints (i.e. more than a feature selected in a set of alternative features or the user attempts to select a feature with an avoid constraint)).
3) Steps 1 and 2 are repeated until every features/concepts, required by the user are selected/deselected or an inconsistency is detected. In this latter case, the selection engine cannot go further and it launches an exception. Such exception describes the OWL individual, which has generated the inconsistency, and the SWRL rule not verified. The feature model has to be modified by the user in order to solve all the conflicts.
4) The user asks to end the selection procedure.
5) The SelectionEngine selects all the default features not already selected, removes all the features that do not have their concept selected and calculates a set of selected features.
6) The user can get only the leaf features selected, all the selected features (leaf + node features) or all the entities selected (leaf feature + node feature + concepts). This last option can be used to obtain an SVG diagram from a feature configuration.

VI. Concluding Remarks

The flexibility of this features model, makes it suitable for modeling a wide range of applications: media processing (images, video, audio) [9], [10], [11], multimedia retrieval tools [12], medical applications [13], [14], [15], [16], [17], [18], [19], [20], biometric applications like [21], object detection, tracking and classification applications [22], [23], [24], specific image processing methods [25].

We plan as a future development to integrate this project with a CASE tool to create and modify graphically a feature model or a configuration. Small improvements could be: adding structural inconsistency checks that cannot be performed by SWRL rules according to the Open World Assumption, i.e. a node can have only a parent node; managing temporal feature constraints next and previous (see Section III-A), in order to create an ordered list of selected features instead of a set (see Section V); improving the SVGDiagramBuilder module with a graphical and textual representation of features constraints.

References

[1] P.-Y. Schobbens, P. Heymans, J.-C. Trigaux, and Y. Bontemps, “Generic semantics of feature diagrams,” *Comput. Netw.*, vol. 51, no. 2, pp.

456–479, Feb. 2007. [Online]. Available: http://dx.doi.org/10.1016/j.comnet.2006.08.008

[2] K. Kang, S. Cohen, J. Hess, W. Nowak, and S. Peterson, *Feature-Oriented Domain Analysis (FODA) Feasibility Study*, 1990.

[3] D. Batory, "Feature models, grammars, and propositional formulas." Springer, 2005, pp. 7–20.

[4] D. Benavides, P. Trinidad, and A. Ruiz-Cortes, "Automated reasoning on feature models," in *LNCS, ADVANCED INFORMATION SYSTEMS ENGINEERING: 17TH INTERNATIONAL CONFERENCE, CAISE 2005*. Springer, 2005, p. 2005.

[5] L. A. Zaid, G. Houben, O. D. Troyer, and F. Kleinermann, "An owl-based approach for integration in collaborative feature modelling," in *SWESE 2008, 4th Workshop on Semantic Web Enabled Software Engineering, workshop at ISWC 2008, the International Semantic Web Conference 2008*, Karslruhe, Germany, Oct. 2008, pp. 93–100.

[6] H. Wang, Y. F. Li, J. Sun, H. Zhang, and J. Pan, "A semantic web approach to feature modeling and verification," in *In Workshop on Semantic Web Enabled Software Engineering (SWESE05*, 2005.

[7] K. Czarnecki and U. W. Eisenecker, *Generative programming: methods, tools, and applications*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000.

[8] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical owl-dl reasoner," *Web Semant.*, vol. 5, no. 2, pp. 51–53, June 2007. [Online]. Available: http://dx.doi.org/10.1016/j.websem.2007.03.004

[9] H. Bannour and C. Hudelot, "Towards ontologies for image interpretation and annotation," in *Content-Based Multimedia Indexing (CBMI), 2011 9th International Workshop on*, june 2011, pp. 211 –216.

[10] T. Perperis, T. Giannakopoulos, A. Makris, D. I. Kosmopoulos, S. Tsekeridou, S. J. Perantonis, and S. Theodoridis, "Multimodal and ontology-based fusion approaches of audio and visual processing for violence detection in movies," *Expert Systems with Applications*, vol. 38, no. 11, pp. 14 102 – 14 116, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957417411007469

[11] K. Shirahama and K. Uehara, "Effectiveness of video ontology in query by example approach," in *Active Media Technology*, ser. Lecture Notes in Computer Science, N. Zhong, V. Callaghan, A. Ghorbani, and B. Hu, Eds. Springer Berlin / Heidelberg, 2011, vol. 6890, pp. 49–58.

[12] D. Giordano, I. Kavasidis, C. Pino, and C. Spampinato, "A semantic-based and adaptive architecture for automatic multimedia retrieval composition," in *Content-Based Multimedia Indexing (CBMI), 2011 9th International Workshop on*, june 2011, pp. 181 –186.

[13] D. Giordano, C. Spampinato, G. Scarciofalo, and R. Leonardi, "An automatic system for skeletal bone age measurement by robust processing of carpal and epiphysial/metaphysial bones," *Instrumentation and Measurement, IEEE Transactions on*, vol. 59, no. 10, pp. 2539 –2553, oct. 2010.

[14] A. Faro, D. Giordano, C. Spampinato, and M. Pennisi, "Statistical texture analysis of mri images to classify patients affected by multiple sclerosis," in *XII Mediterranean Conference on Medical and Biological Engineering and Computing 2010*, ser. IFMBE Proceedings, P. D. Bamidis, N. Pallikarakis, and R. Magjarevic, Eds. Springer Berlin Heidelberg, 2010, vol. 29, pp. 272–275.

[15] A. Faro, D. Giordano, I. Kavasidis, C. Pino, C. Spampinato, M. G. Cantone, G. Lanza, and M. Pennisi, "An interactive tool for customizing clinical transacranial magnetic stimulation (tms) experiments," in *XII Mediterranean Conference on Medical and Biological Engineering and Computing 2010*, ser. IFMBE Proceedings, P. D. Bamidis, N. Pallikarakis, and R. Magjarevic, Eds. Springer Berlin Heidelberg, 2010, vol. 29, pp. 200–203.

[16] A. Faro, D. Giordano, C. Spampinato, S. Ullo, and A. Di Stefano, "Basal ganglia activity measurement by automatic 3-d striatum segmentation in spect images," *Instrumentation and Measurement, IEEE Transactions on*, vol. 60, no. 10, pp. 3269 –3280, oct. 2011.

[17] A. Faro, D. Giordano, C. Spampinato, D. De Tommaso, and S. Ullo, "An interactive interface for remote administration of clinical tests based on eye tracking," in *Proceedings of the 2010 Symposium on Eye-Tracking Research & Applications*, ser. ETRA '10. New York, NY, USA: ACM, 2010, pp. 69–72. [Online]. Available: http://doi.acm.org/10.1145/1743666.1743683

[18] D. Giordano, R. Leonardi, F. Maiorana, G. Scarciofalo, and C. Spampinato, "Epiphysis and metaphysis extraction and classification by adaptive thresholding and dog filtering for automated skeletal bone age analysis," in *Engineering in Medicine and Biology Society, 2007. EMBS 2007. 29th Annual International Conference of the IEEE*, aug. 2007, pp. 6551 –6556.

[19] A. Faro, D. Giordano, C. Pino, and C. Spampinato, "Visual attention for implicit relevance feedback in a content based image retrieval," in *Proceedings of the 2010 Symposium on Eye-Tracking Research and Applications*, ser. ETRA '10. New York, NY, USA: ACM, 2010, pp. 73–76. [Online]. Available: http://doi.acm.org/10.1145/1743666.1743684

[20] A. Faro, D. Giordano, F. Maiorana, and C. Spampinato, "Discovering genes-diseases associations from specialized literature using the grid," *Information Technology in Biomedicine, IEEE Transactions on*, vol. 13, no. 4, pp. 554 –560, july 2009.

[21] A. Faro, D. Giordano, and C. Spampinato, "An automated tool for face recognition using visual attention and active shape models analysis," in *Engineering in Medicine and Biology Society, 2006. EMBS '06. 28th Annual International Conference of the IEEE*, 30 2006-sept. 3 2006, pp. 4848 –4852.

[22] C. Spampinato, "Adaptive objects tracking by using statistical features shape modeling and histogram analysis," in *Advances in Pattern Recognition, 2009. ICAPR '09. Seventh International Conference on*, feb. 2009, pp. 270 –273.

[23] C. Spampinato, D. Giordano, R. Di Salvo, Y.-H. J. Chen-Burger, R. B. Fisher, and G. Nadarajan, "Automatic fish classification for underwater species behavior understanding," in *Proceedings of the first ACM international workshop on Analysis and retrieval of tracked events and motion in imagery streams*, ser. ARTEMIS '10. New York, NY, USA: ACM, 2010, pp. 45–50. [Online]. Available: http://doi.acm.org/10.1145/1877868.1877881

[24] C. Spampinato, Y.-H. Chen-Burger, G. Nadarajan, and R. B. Fisher, "Detecting, tracking and counting fish in low quality unconstrained underwater videos." in *VISAPP (2)*, A. Ranchordas and H. Arajo, Eds. INSTICC - Institute for Systems and Technologies of Information, Control and Communication, 2008, pp. 514–519. [Online]. Available: http://dblp.uni-trier.de/db/conf/visapp/visapp2008-2.html#SpampinatoCNF08

[25] F. Cannavo, G. Nunnari, D. Giordano, and C. Spampinato, "Variational method for image denoising by distributed genetic algorithms on grid environment," in *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2006. WETICE '06. 15th IEEE International Workshops on*, june 2006, pp. 227 –232.