

AIM@SHAPE

Advanced and Innovative Models And Tools for the
development of Semantic-based systems for
Handling, Acquiring, and Processing knowledge
Embedded in multidimensional digital objects

IST NoE No 506766

Deliverable D1.5.1



Report on an integrated view of the domain ontologies – 2nd version

Circulation:	¹ CO
Partner(s):	ITI, IMATI, DISI, EPFL, IGD, MPII, INRIA
Authors:	G. Vasilakis, M. Pitikakis, C. Catalano, W. Saleem, L. Saboret, L. Papaleo, A. Rojas, N. Sevilmis
Version:	03
Stage:	100%
Date:	Monday, November 13, 2006

¹ Please indicate the dissemination level using one of the following codes:

PU = Public

PP = Restricted to other programme participants (including the Commission Services).

RE = Restricted to a group specified by the consortium (including the Commission Services).

CO = Confidential, only for members of the consortium (including the Commission Services).

Copyright

© Copyright 2006 The AIM@SHAPE Consortium

consisting of:

CNR-IMATI-GE	C.N.R. – Istituto di Matematica Applicata e Tecnologie Informatiche Dept. of Genova, Italy
DISI	Università di Genova – Dipartimento di Informatica e Scienze dell'Informazione, Italy
EPFL	École Polytechnique Federale de Lausanne, Switzerland
FhG/IGD	Fraunhofer Institut für Graphische Datenverarbeitung, Germany
INPG	Institut National Polytechnique de Grenoble, France
INRIA	Institut National de Recherche en Informatique et Automatique, France
ITI-CERTH	Informatics and Telematics Institut – Center for Research and Technology Hellas, Greece
UNIGE	Université de Genève, Switzerland
MPII	Max-Planck-Institut für Informatik, Germany
SINTEF	Stiftelsen for industriell og teknisk forskning ved Norges Tekniske Høgskole, Norway
TECHNION	Technion – Israel Institute of Technology, Israel
UU	Utrecht University, Netherlands
WEIZMANN	Weizmann Institute of Science, Israel

This document may not be copied, reproduced, or modified in whole or in part for any purpose without written permission from the AIM@SHAPE Consortium. In addition to such written permission to copy, reproduce, or modify this document in whole or part, an acknowledgement of the authors of the document and all applicable portions of the copyright notice must be clearly referenced.

All rights reserved.

This document may change without notice.

Document History

Vers.	Issue Date	Stage	Content and changes
1	02 October 2006	20%	First draft
2	18 October 2006	80%	Second draft
3	5 November 2006	100%	Final version

Executive Summary

This document contains a description of the deliverable D1.5.1 of the IST NoE AIM@SHAPE. The deliverable *D1.5.1 Report on an integrated view of the domain ontologies – 2nd version* is intended to provide an integrated view of the ontology development process in the network.

The task leader is ITI-CERTH and is actively supported by all other AIM@SHAPE partners. The document is structured as follows.

In section 1 an introduction to the evolution of the common ontologies is given, describing also their integration with the domain ontologies, the Shape Repository and the Tool Repository.

In section 2 the evolution of the Shape Common Ontology is described from version 1 to the current version 2. The modifications that have been made are described in detail as well as some justification for applying the changes.

In section 3 the evolution of the Tool Common Ontology is described from version 1 to the current version 2. The modifications that have been made are described in detail as well as some justification for applying the changes.

In section 4 some concluding remarks are given, summarizing the evolution of the common ontologies and their significance within the ontology development in the network and the Digital Shape Workbench as well.

Table of Contents

1	INTRODUCTION.....	5
1.1	INTEGRATING THE DOMAIN ONTOLOGIES AND THE REPOSITORIES	5
1.2	EVOLUTION STEPS AND FORMAL CQS	6
2	THE COMMON ONTOLOGY FOR SHAPES	7
3	THE COMMON ONTOLOGY FOR TOOLS	14
4	CONCLUDING REMARKS	20
4.1	SUMMARY OF CHANGES FROM THE PREVIOUS VERSION.....	20

1 INTRODUCTION

In the first version of this deliverable an introduction to the ontology development in the network was given, defining the various domains of interest and explaining the reasons which led to the creation of the common ontologies for shapes and tools. These common ontologies constitute higher-level ontologies and can be viewed as an attempt to record information that is relevant for shapes and tools and is required for managing such resources within the associated repositories.

This second version of the deliverable mainly presents the evolution of the common ontologies as these essentially represent the effort to integrate the development in the domain ontologies under a common ontological framework. This effort began almost a year ago and has now reached a level of maturity that enables us to focus more on developing the domain ontologies, whose progress has been necessarily slowed down to allow for the validation of the common ontologies to take place.

The creation and validation of the common ontologies has taken a lot of effort and resources. Some changes and restructurings produced by the validation process may seem small but are in fact subtle and are the result of several sessions, meetings and email exchanges, which were required to resolve the issues presented. A notable example is the restructuring of the *Shape Representation* hierarchy in the common ontology for shapes. This hierarchy has been restructured numerous times in order to resolve disagreements in the correct way of capturing the implicit relations between several shape categories. This is of course an integral part of ontology development and shows the difficulty in combining different backgrounds of expertise in order to model in a standard and agreed manner knowledge that is shared across multiple domains.

1.1 Integrating the domain ontologies and the repositories

At this stage of development the domain ontologies have been fully integrated with the common ontologies. This means that there is no duplication of knowledge or redundancy in the common ontologies as was the case in their previous release. The domain ontologies have been carefully refactored to use and extend the common ontologies, eliminating redundancy and promoting reusability.

Besides the integration of common knowledge captured in the domain ontologies, another requirement for the common ontologies has been to include the information present in the Shape and Tool repositories. This is a fundamental requirement as the preservation of the structure and functionality of these repositories is of utmost importance. This has been achieved and future restructurings of the common ontologies will be automatically reflected on the structure of the repositories. This simply means that after the integration of the common ontologies with the repositories, the structure of the latter is ontology driven and follows that of the common ontologies in particular.

Full integration with the domain ontologies and with the information in the repositories does not imply that the common ontologies will not be modified over time. Further changes, however, will be made through a formal process of validation which involves a hierarchy of responsibilities associated with partners who are experts in the various

domains that are being modeled, as well as the partners managing the Shape and Tool repositories and the WP1 leader who is overall responsible for the development of the ontologies in the network. This procedure is established in order to guarantee that changes, mostly, but not necessarily exclusively, in the common ontologies must be submitted by the partner that proposes them and filtered through a channel involving all the partners who should be aware of and validate these changes. This marks the central role of the common ontologies not just in ontology development but also with regards to the Digital Shape Workbench as well.

1.2 Evolution steps and formal CQs

In the previous version of this document we introduced the notion of an *evolution step* in order to impose a more rigorous evaluation procedure on the ontology development process. An evolution step signifies a well-defined progress of an ontology by showing that it takes a concrete step towards completeness by capturing a richer set of CQs.

The introduction of evolution steps in the ontology development process has been applied with success in the previous 4-month period and partners have responded well. Safe conclusions cannot be reached at this point, however, we feel that evolution steps are likely to help in keeping the ontology development in accordance with the methodological approach that has been specified and make it less based on mere intuition.

To take this one step further, the need for a formal way to describe CQs has been justified in order to effectively provide a qualitative measure of the ontology development process. CQs will be specified in a formal manner in every evolution step using *nRQL* (new Racer Query Language). The nRQL definition for a CQ can be retrieved when posing the query using the Search Engine prototype. The Search Engine first creates the query in nRQL and it can be easily retrieved in the nRQL format before submission. This will ensure that CQs are properly defined, are logically consistent and unambiguous.

2 THE COMMON ONTOLOGY FOR SHAPES

An overview of the first version of the common ontology for shapes that was described in the previous version of this document is shown in Figure 1.

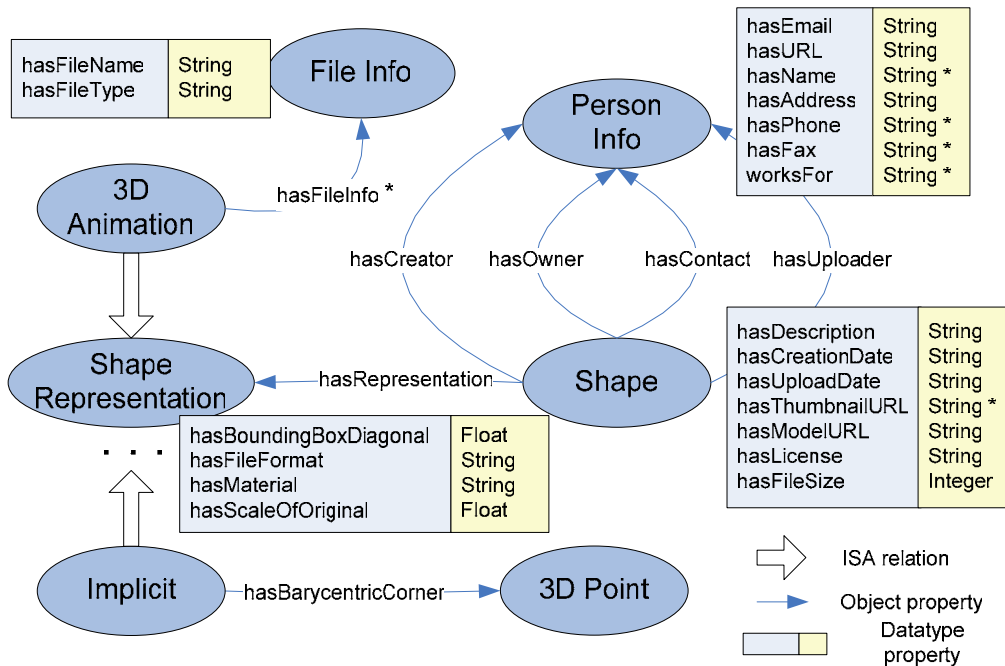


Figure 1: Shape common ontology (1st version).

This structure has undergone a lot of modifications which will be described in detail next. In this first version the central classes in the ontology were *Shape* and *Shape Representation*. The reason why we needed two classes for the representation of a shape was to store the more general metadata in one class (*Shape*) and put the metadata related to the representation of the shape in the other (*Shape Representation*). This has the advantage that the *Shape Representation* class is more reusable as it was intended to capture metadata that could be shared by two or more *Shape* objects.

In the second version of the ontology the class *Shape* was removed. The reason is that the class *Shape Representation* was not as reusable as we had hoped, nor was there any realistic chance that it could be with a few modifications. Therefore, there was no need to complicate the structure by having a redundant class. Most of the datatype properties in *Shape* (like *hasDescription*, *hasCreationDate*, *hasUploadDate*, etc.,) have migrated to *Shape Representation*.

In the first version of the ontology class *File Info* was associated only with class *3DAnimation*, which is a subclass of *Shape Representation*. In the second version *File Info* is a much more general class and is associated with *Shape Representation*. This makes more sense since there were already some properties in class *Shape* (i.e. *hasFileSize*, and *hasFileURL*) which were more general and could migrate to *File Info*. This is what happened in the second version of the ontology where class *Shape*

Representation is linked with *File Info* and some properties that were originally in class *Shape* moved to *File Info*.

Class *3DPoint* originally found in the first version of the ontology has been removed in the second version as it is not used anymore. A new class, *Structural Descriptor*, has been added and is linked to *Shape Representation*.

Another new class, *Institution Info*, has been added in the second version of the ontology and represents information about various institutions that may be associated with a shape. *Shape Representation* is associated with classes *Person Info* and *Institution Info* through four different object properties: *hasCreator*, *hasOwner*, *hasContact* and *hasUploader* (Figure 2). In other words the representation of a person or an institute is related with the representation of a shape through four distinct roles: as a creator, as an owner, as a contact person, or as the person who uploaded the shape. The fact that either *Person Info* or *Institution Info* can play these roles is specified as the union of these two classes in the ontology. Furthermore, there is a relation *worksFor* between *Person Info* and *Institution Info*. This replaces the property *worksFor* that originally belonged to class *Shape* in the first version of the ontology.

The last modifications with regards to the first class level of the ontology involve the addition of two more classes, namely *Shape Group* and *Sub Group*. These classes were added in order to support the feature of groups in the Shape Repository. Shapes are grouped when they have something in common. Most of the times we group copies of a shape that differ in some key features like their resolution, for example. Within a group of shapes we can have several subgroups and a representative shape. The representative shape is essentially the original shape whose variations are grouped in the various subgroups. Each subgroup contains variations of the representative shape that differ in one key feature.

An overview of the second version of the ontology for shapes is shown in Figure2.

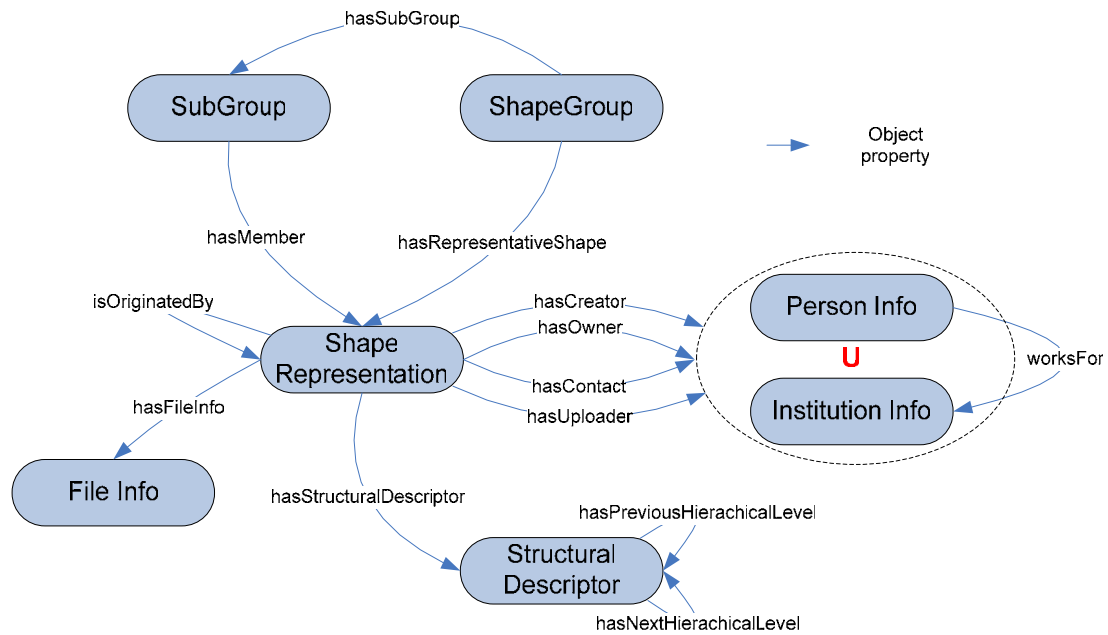


Figure 2: Shape common ontology (2nd version).

The structure of the ontology as can be seen is still fairly simple. The ontology includes 15 object properties of which 13 are among 7 level-one classes. At a quick glance the new structure seems more concise, more natural and more functional, while at the same time simplicity is preserved – as it should.

A full account of the first class level properties for the second version of the ontology is given in Figure 3. In this and the following diagrams displaying class properties, a property type with an asterisk denotes multiple occurrence of the property while absence of the asterisk denotes single occurrence.

File Info		Institution Info		Shape Representation	
hasFileFormat	string	hasAddress	string	hasAvailabilityLevel	string
hasFileName	string	hasName	string *	hasContentType	string *
hasFileSize	integer	hasShortName	string	hasCopyright	string
hasFileURL	string *	hasURL	string *	hasCreationDate	string
hasPurpose	string			hasDescription	string
				hasGalleryImageURL	string *
				hasKeyword	string *
				hasLastModificationDate	string
				hasLicense	string
				hasModelURL	string
				hasName	string
				hasOrigin	string
				hasQuality	integer
				hasRelatedResource	string *
				hasStatus	string
				hasSynthesisDescription	string
				hasThumbnailImageURL	string
				hasUploadDate	string

Shape Group		Person Info	
hasDescription	string	hasAddress	string
hasName	string	hasEmail	string *
		hasFax	string
		hasName	string *
		hasPhone	string
		hasTitle	string *
		hasURL	string *

Sub Group	
hasDescription	string
hasLevelNumber	integer

Figure 3: Shape common ontology (2nd version) – First level class properties.

The *Shape Representation* hierarchy for the first version of the ontology is shown in Figure 4. Different class levels are painted in different colour in the diagram. First level classes are shown in blue colour, second level classes are shown in light blue colour and third level classes are shown in yellow colour in the hierarchy.

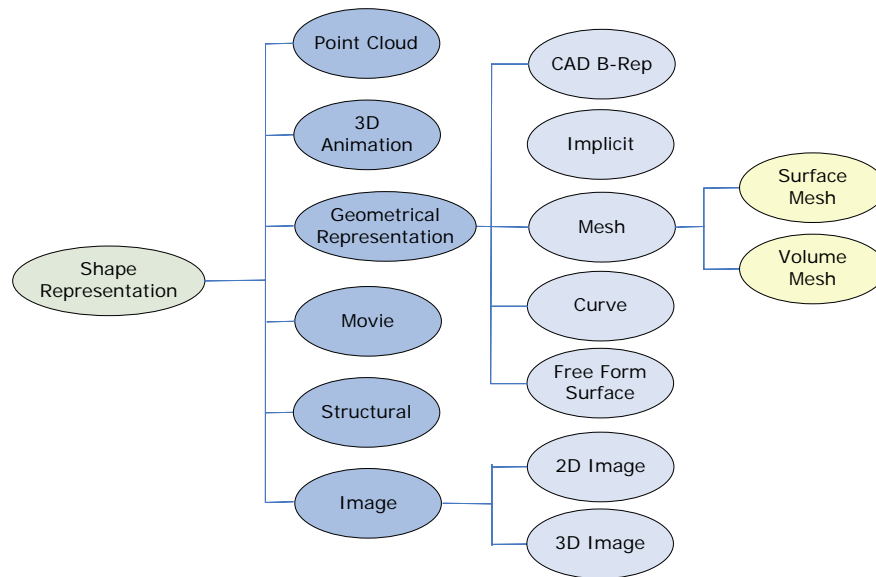


Figure 4: Shape common ontology (1st version) – Shape Representation hierarchy.

The same hierarchy for the second version of the ontology is depicted in Figure 5. It can easily be seen that the new structure is much more extensive and contains a lot of specialized classes that were not present in the first version of the ontology.

This is expected as the structure of the common ontology for shapes (*SCO*) is also shared with the structure of the Shape Repository. This means that the structure for the *SCO* should at the very least contain enough information to differentiate between all types of shapes already existing in the Shape Repository. So the most important requirement currently is full integration with the Shape Repository. Obviously, another requirement is to factor out common knowledge needed by the domain ontologies and store it in the *SCO* in order to be shared and easily reused. The first requirement stated is straightforward as the number and the type of shapes that are currently stored in the Shape Repository can be easily ascertained.

The second requirement however is more challenging and could result in making the ontology difficult to maintain and reuse. The danger here is to include a lot of information which is not actually common among the domain ontologies. This would make the ontology unnecessarily large, which would have significant impact in performance (more so because all domain ontologies import it) and also make its reuse harder. Furthermore, the ontology would also become harder to maintain.

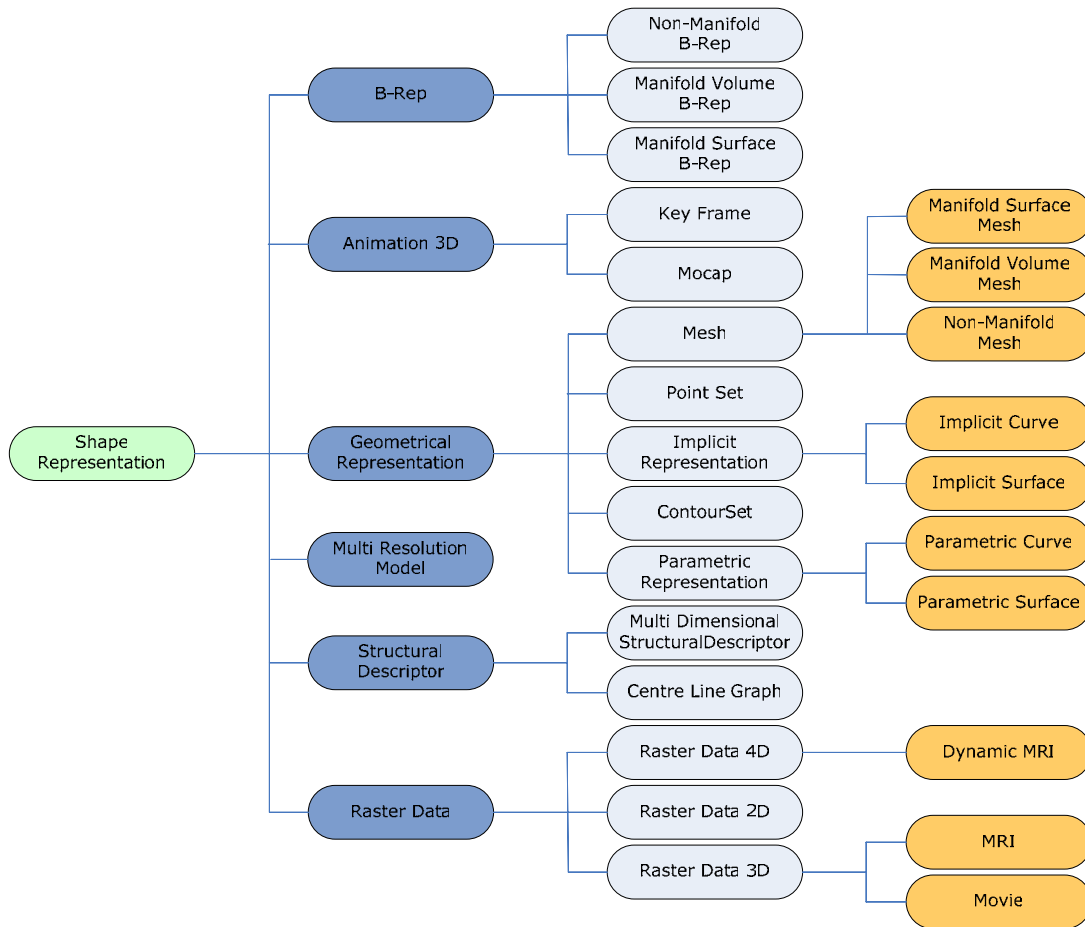


Figure 5: Shape common ontology (2nd version) – Shape Representation hierarchy.

To clarify this last point let's consider the scenario of addressing a change that requires to insert new knowledge into some ontology. If this change concerns knowledge that we need to add to a domain ontology, then validation and consulting sessions are needed within the cluster working only on that ontology. If, however, we had decided to make this knowledge common and insert it in one of the common ontologies, then validation would involve all clusters working on the domain ontologies and also the partners responsible for the repository that might be affected by the change. In the second case, applying modifications and maintenance in general is much more costly.

Of course this does not imply that the common ontology will remain static and will not evolve over time. This is unrealistic as it is definitely possible that a shape of a new type, possibly some specialization of the classes in the *Shape Representation* hierarchy which had not been taken into account, might need to be inserted in the Shape Repository and the SCO will need to be modified to accommodate this new type.

To emphasize this last point, there are some classes in the ontology that still do not have any properties defined at this stage. This means that there are metadata specific to these classes, so a distinction between them has been considered necessary, and these will be specified in the next version of the ontology. The classes mentioned are:

- *Contour Set* under *Geometrical Representation*;
- *Non Manifold B-Rep* under *B-Rep*;
- *Manifold Surface B-Rep* under *B-Rep*;
- *Manifold Volume B-Rep* under *B-Rep*;
- *Raster Data 4D* under *Raster Data*.

The arguments that have been mentioned regarding the reusability of the common ontology for shapes also apply to the common ontology for tools which is described in the following section.

The datatype properties of the second level classes in the second version of the common ontology for shapes are described in Figure 6.

Structural Descriptor		Animation 3D		B-Rep	
hasAttributeDescription	string	hasAnimationFormat	string	hasAccuracyOfConnectionBetweenFaces	float
hasCreationMethod	string	hasDuration	float	hasCurvilinearEdges	boolean
hasNumberOfConnectedComponents	integer	hasFrameRate	integer	hasDimension	integer
hasType	string	hasGeometryFormat	string	hasGenus	integer
isAttributed	boolean	hasStructureType	string	hasMaxContinuityAcrossFaces	string
		hasTextureFormat	string	hasMinContinuityOfTheFaces	string
				hasNumberOfConnectedComponents	integer
				hasNumberOfFaces	integer
				hasNumberOfTrimmedFaces	integer
				hasOnlyAnalyticalSurfaces	boolean
				hasTextures	boolean
				hasType	string
				isSelfIntersecting	boolean
				isTopologicalComplex	boolean

Geometrical Representation		Raster Data	
hasBoundingBoxDiagonal	float	hasDimensionX	integer
hasMaterial	string	hasMeasureType	string
hasScaleOfOriginal	float		

Figure 6: Shape common ontology (2nd version) – Second level class properties.

The datatype properties of the third level classes in the second version of the common ontology for shapes are described in Figure 7.

Implicit Representation		Key Frame		Parametric Representation	
hasApproximationTolerance	float	hasInterpolatorType	string	hasDegree	integer
hasDegree	integer	hasNumberOfKeyFrames	integer	hasNumberOfControlPoints	integer
hasGenus	integer			hasPolynomialModelType	string
hasImplicitBasis	string	Mocap		hasType	string
hasParametricDescriptionURL	string	hasFiltering	string	hasTypeOfParametricDomain	string
hasType	string	isRawData	boolean	isClosed	boolean
isGeneratedFromParametricDescription	string			isSelfIntersecting	boolean
isSelfIntersecting	boolean				

Mesh		Point Set		Centre Line Graph	
hasNumberOfConnectedComponents	integer	hasDigitalImages	string	hasNumberOfArcs	integer
hasNumberOfEdges	integer	hasGeoReference	string	hasNumberOfNodes	integer
hasNumberOfFaces	integer	hasNumberOfPoints	integer	isAcyclic	boolean
hasNumberOfVertices	integer	hasNumberOfScans	integer	isDirected	boolean
hasType	string	hasPointInfo	string *		
		hasType	string		

Multi Dimensional Structural Descriptor		Raster Data 2D		Raster Data 3D	
hasMaximumDimension	integer	hasDimensionY	integer	hasDimensionY	integer
isSimplyConnected	boolean			hasThickness	float

Figure 7: Shape common ontology (2nd version) – Third level class properties.

Lastly, the datatype properties of the fourth level classes in the second version of the common ontology for shapes are described in Figure 8.

Non Manifold Mesh		Implicit Curve		MRI	
hasNumberOf1ConnectedComponents	integer	hasBoundingBox	boolean	hasAcquisitionDate	string
hasNumberOf2Cycles	integer	hasContinuity	string	hasAcquisitionMatrix	string
hasNumberOfCells	integer	isClosed	boolean	hasAcquisitionTime	string
hasNumberOfConnectedSimplexesOfDim1	integer	isPlanar	boolean	hasBitStored	integer
hasNumberOfConnectedSimplexesOfDim2	integer			hasDimensionZ	integer
hasNumberOfNonManifoldEdges	integer			hasEchoTime	float
hasNumberOfNonManifoldVertices	integer			hasFlipAngle	float
hasNumberOfWireEdges	integer			hasImageOrientation	string
hasTopologicalDescription	string			hasImagePosition	string
				hasMagneticFieldStrength	float
				hasMRIAcquisitionType	string
				hasMRIModality	string
				hasPatientAge	integer
				hasPatientSex	string
				hasPatientWeight	float
				hasPixelSpacing	float
				hasReceivingCoil	string
				hasRepetitionTime	float
				hasSliceThickness	float
Dynamic MRI		Parametric Surface			
hasAcquisitionDate	string	hasContinuity	string		
hasAcquisitionFrequency	string	hasGenus	integer		
hasAcquisitionMatrix	string	hasMaxDegree	string		
hasAcquisitionTime	string	Polynomial			
hasBitStored	integer	isTrimmed	boolean		
hasDimensionZ	integer				
hasEchoTime	float	Parametric Curve			
hasFlipAngle	float	hasContinuity	string		
hasImageOrientation	string	isPlanar	boolean		
hasImagePosition	string				
hasMagneticFieldStrength	float	Movie			
hasMRIAcquisitionType	string	hasCodec	string *		
hasMRIModality	string	hasDimensionT	integer		
hasNumberOfAcquisitionPlane	integer	hasDuration	float		
hasNumberOfTemporalPositions	integer	hasFrameRate	integer		
hasPatientAge	integer	hasSound	boolean		
hasPatientSex	string	hasStreaming	boolean		
hasPatientWeight	float	hasUncompressed	integer		
hasPixelSpacing	float	FileSize			
hasReceivingCoil	string	Implicit Curve			
hasRepetitionTime	float	hasBoundingBox	boolean		
hasSliceThickness	float	hasContinuity	string		
		hasNumberOfSingularities	integer		
		isClosed	boolean		

Figure 8: Shape common ontology (2nd version) – Fourth level class properties.

3 THE COMMON ONTOLOGY FOR TOOLS

An overview of the first version of the common ontology for tools that was described in the previous version of this document is shown in Figure 9.

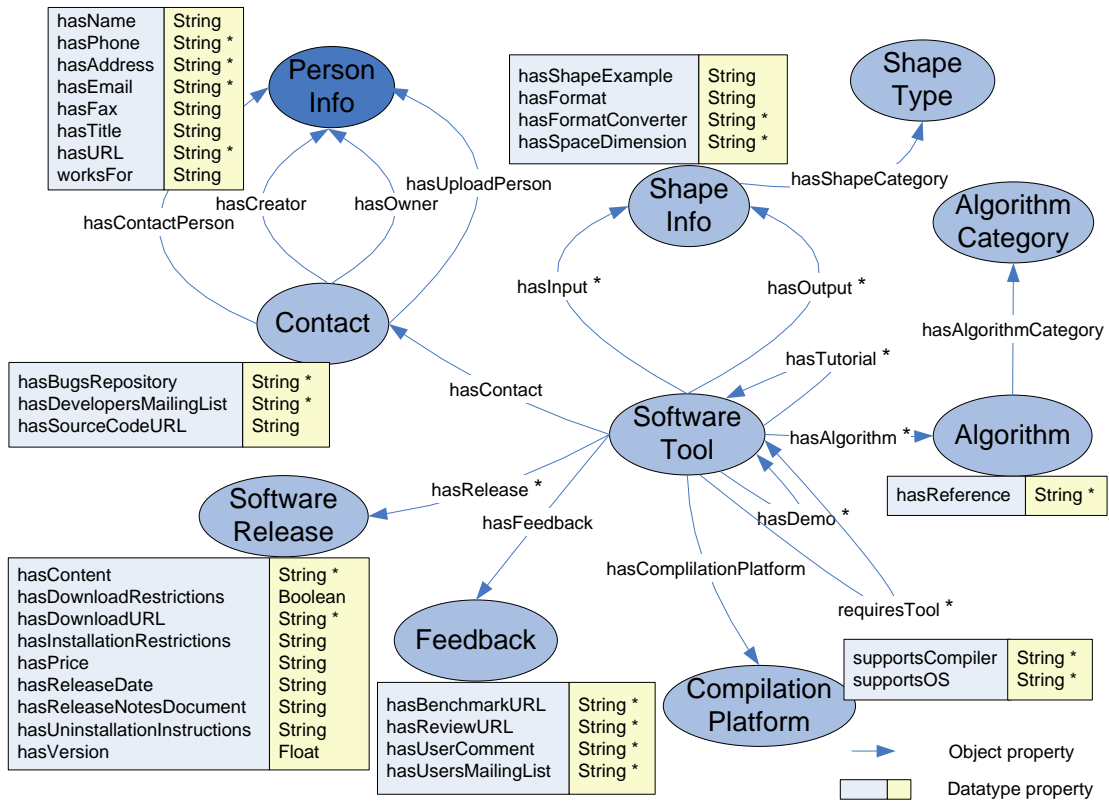


Figure 9: Tool common ontology (1st version).

This first level class structure has not been affected very much through the evolution of the ontology to its second version. In fact, the first level classes have been reduced so that classes *Contact* and *Feedback* have been removed altogether. *Contact* was used in order to make *Person Info* more reusable (as was the case of class *Shape* in the first version of the common ontology for shapes). However, this did not prove to be the case and slightly complicated the ontology. On the other hand, class *Feedback* did not have a clear role in the structure so it was removed and all of its datatype properties have migrated to class *Software Tool*.

Class *Algorithm Category* has been renamed to *Functionality* and its role has been refined in order to allow relations with class *Software Tool* as well. All these changes are reflected in Figure 10.

between the two structures and has required a lot of effort for experts in the various related disciplines in order to converge and reach consensus. Therefore, to develop two distinct such hierarchies would have been time consuming and inefficient, at best. The original Shape Type hierarchy in the first version of the Tool Common Ontology is shown in Figure 12.

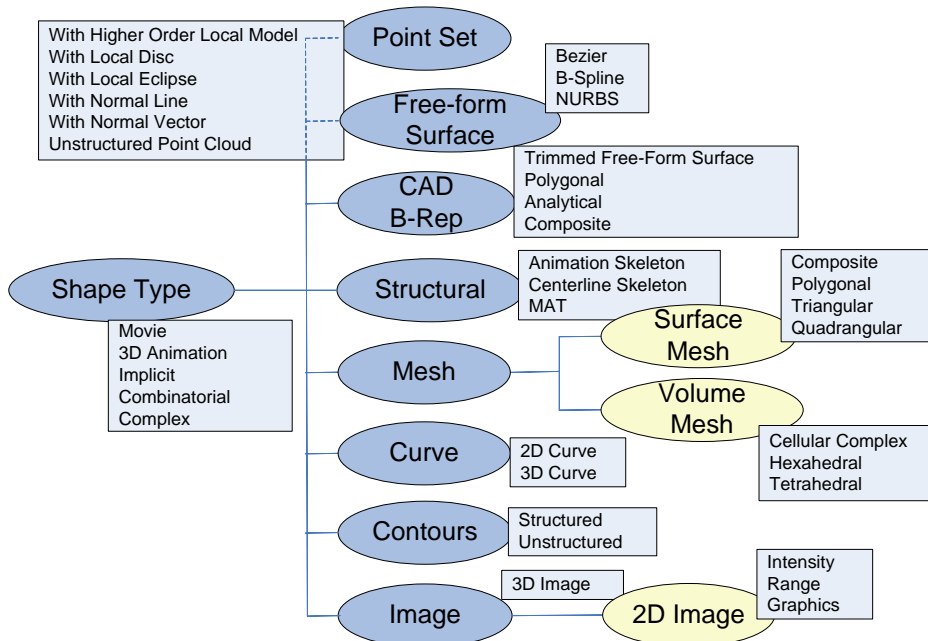


Figure 12: Tool common ontology (1st version) – Shape Type hierarchy.

In this diagram the instances of the various classes in the hierarchy are shown in a light blue background. We can compare this structure with the one found in the new version of the ontology which is shown in Figure 13. The instances are not shown in this figure but it is clear that the structure is a lot more detailed and refined. We can also compare it with the structure of the *Shape Representation* hierarchy in the Shape Common Ontology which is depicted in Figure 5. Essentially, the new *Shape Type* hierarchy in the tools common ontology is a slightly simplified version of the new *Shape Representation* hierarchy in the Shape Common Ontology.

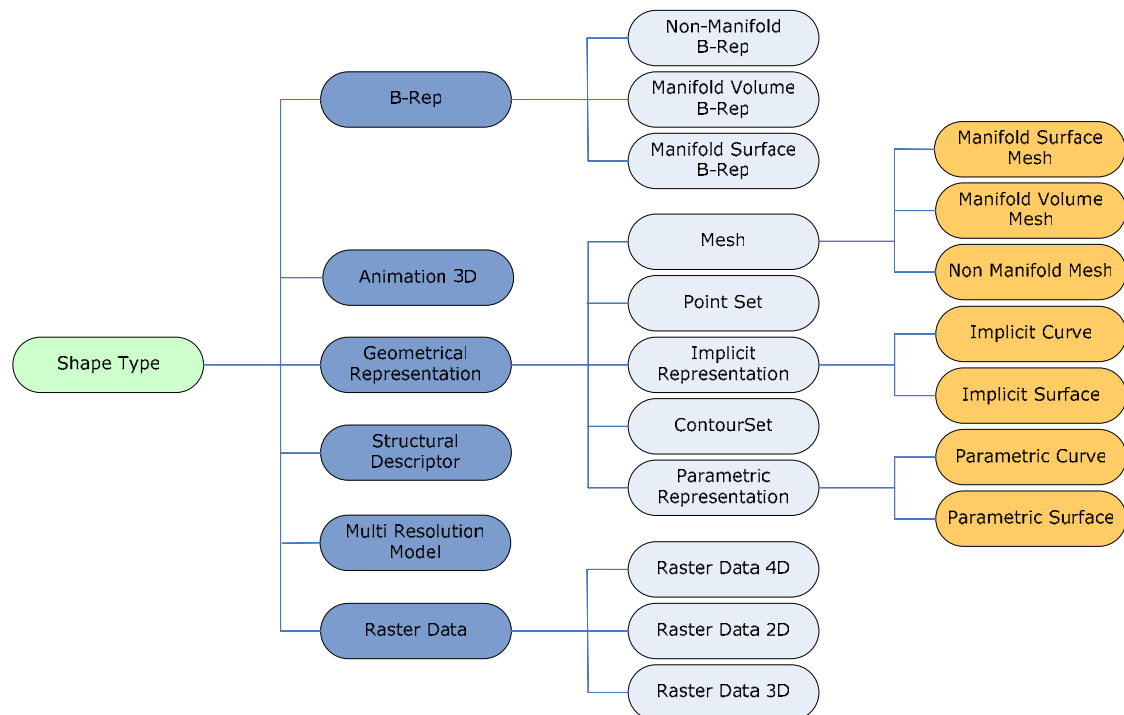


Figure 13: Tool common ontology (2nd version) – Shape Type hierarchy.

The first class-level instances of the new *Shape Type* hierarchy are shown in Figure 14. The second class-level instances of the hierarchy are shown in Figure 15, while the third class-level instances of the hierarchy are shown in Figure 16.

These instances show the level of detail that we can use in specifying the type of the shape a software tool is able to take as input or produce as output. This extensive list of instances obviously cannot be compared with the small number of instances shown in Figure 12 that were present in the previous version of the Tool Common Ontology.

An instance essentially is a leaf in the *Shape Type* hierarchy and thus represents the most specific shape type with respect to the path in the hierarchy that has been chosen. For example, *Geometrical Representation* is more general than *Implicit Representation*, which is more general than *Implicit Curve* which in turn is more general than *Bezier Implicit Curve*. Thus, along this path *Bezier Implicit Curve* constitutes the largest degree of specialization possible for defining the type of a specific shape. Clearly, other leaves in the same path (instances of *Implicit Curve* in the hierarchy such as *BSpline Implicit Curve*) would also provide the most specific type information.

However, we may not be interested in defining such a specific type for the shape and specifying a more general class in the hierarchy could instead be sufficient. For every class in the hierarchy there is a special-purpose instance for such a case. This special instance takes the name of the class prefixed by “Other”. Then, for the class *Implicit Representation* this instance is *Other Implicit Representation*. The reason for having these special instances is that it is not feasible to list all the possible types up to the most specific level. Furthermore, even if it were feasible we would probably still need to specify a more general type in some occasions instead of being very specific. The most

straight-forward solution in order to specify a more general type than the one modeled by an instance would be to give the name of its class. However, this is not allowed in OWL because we cannot specify the name of a class in instance-level relations and reason on the hierarchy.

Animation 3D	Structural Descriptor	Multi Resolution Model
KeyFrameAnimation MotionCaptureAnimation OtherAnimation3D	CentreLineGraph MultiDimensionalStructuralDescriptor OtherStructuralDescriptor	MultiTessellationModel OtherMultiResolutionModel
Geometrical Representation	B-Rep	Raster Data
OtherGeometricalRepresentation	OtherBRep	OtherRasterData

Figure 14: Tool common ontology (2nd version) – Shape Type hierarchy first class-level instances.

Raster Data 2D	Manifold Surface B-Rep	Raster Data 3D
BlackWhiteRasterData2D GreyscaleRasterData2D HeightRasterData2D IndexedRasterData2D OtherRasterData2D RGBRasterData2D	ManifoldAnalyticalSurfaceBRep ManifoldImplicitSurfaceBRep ManifoldMixedSurfaceBRep ManifoldParametricSurfaceBRep ManifoldPolygonalSurfaceBRep OtherManifoldSurfaceBRep	BlackWhiteRasterData3D GreyscaleRasterData3D HeightRasterData3D IndexedRasterData3D MagneticResonanceImage Movie OtherRasterData3D RGBRasterData3D XYZRasterData3D
Implicit Representation	Manifold Volume B-Rep	Raster Data 4D
OtherImplicitRepresentation	ManifoldAnalyticalVolumeBRep ManifoldImplicitVolumeBRep ManifoldMixedVolumeBRep ManifoldParametricVolumeBRep ManifoldPolygonalVolumeBRep OtherManifoldVolumeBRep	
Mesh	Non Manifold B-Rep	
OtherMesh		
Parametric Representation		
OtherParametricRepresentation	NonManifoldAnalyticalBRep NonManifoldImplicitBRep NonManifoldMixedBRep NonManifoldParametricBRep NonManifoldPolygonalBRep OtherNonManifoldBRep	
Point Set	Contour Set	
OtherPointSet PointSetWithHigherOrderLocalModels PointSetWithLocalDiscs PointSetWithLocalEllipses PointSetWithNormalDirections PointSetWithOrientedNormals		
	OtherContourSet StructuredContourSet	

Figure 15: Tool common ontology (2nd version) – Shape Type hierarchy second class-level instances.

Manifold Surface Mesh	Manifold Volume Mesh	Non Manifold Mesh
CompositeManifoldSurfaceMesh OtherManifoldSurfaceMesh PolygonalManifoldSurfaceMesh QuadrangularManifoldSurfaceMesh TriangularManifoldSurfaceMesh	HexahedralManifoldVolumeMesh OtherManifoldVolumeMesh TetrahedralManifoldVolumeMesh	CompositeNonManifoldSurfaceMesh HexahedralNonManifoldVolumeMesh OtherNonManifoldMesh PolygonalNonManifoldSurfaceMesh QuadrangularNonManifoldSurfaceMesh TetrahedralNonManifoldVolumeMesh TriangularNonManifoldSurfaceMesh
Parametric Curve	Implicit Curve	Parametric Surface
BezierParametricCurve BSplineParametricCurve NURBSParametricCurve OtherParametricCurve PolynomialParametricCurve RationalParametricCurve	BezierImplicitCurve BSplineImplicitCurve OtherImplicitCurve PolynomialImplicitCurve	
	Implicit Surface	
	BezierImplicitSurface BSplineImplicitSurface OtherImplicitSurface PolynomialImplicitSurface	

Figure 16: Tool common ontology (2nd version) – Shape Type hierarchy third class-level instances.

The *Functionality* hierarchy is used to define what the functionality of an *Algorithm* or a *Software Tool* is. This hierarchy is shown in Figure 17. Again the same modeling technique used for the *Shape Type* hierarchy is applied here. Instances of the hierarchy are not shown as there are far too many.

A final modification that has been included in the second version of the ontology is the addition of a new subclass of the class *Software Tool*. The name of the new class is *Independent Application* and was necessary in order to be fully compliant with the needs of the Tool Repository. As with the Shape Repository and the SCO, the Tool Repository is now fully consistent with the Tool Common Ontology and follows an ontology driven structure.

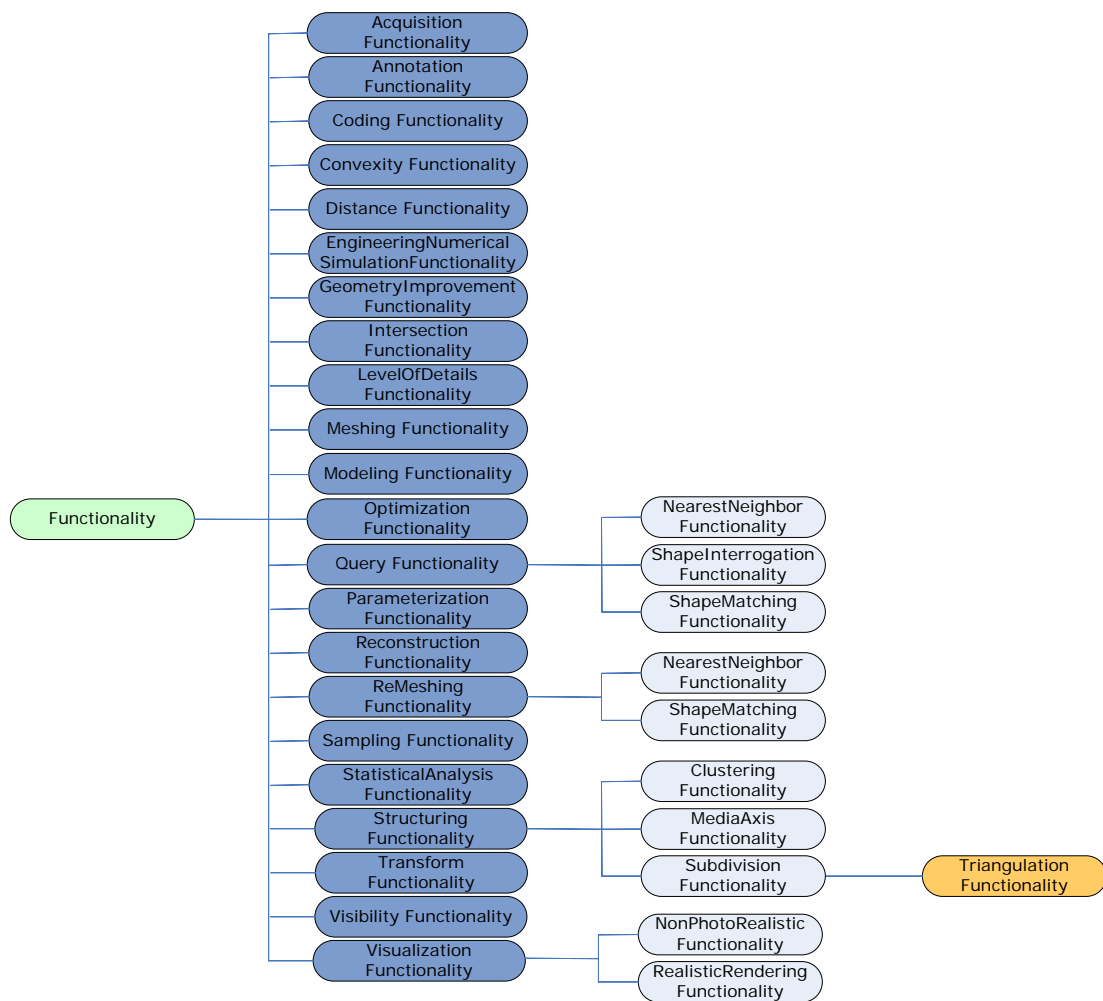


Figure 17: Tool common ontology (2nd version) – Functionality hierarchy.

4 CONCLUDING REMARKS

In this deliverable we presented the evolution of the common ontologies for shapes and tools from their first version, as was described in the previous version of this document, to their current second version.

A large part of the ontology development effort in the past year has involved the common ontologies. Due to their central role both in ontology development but also with regards to the structure of the Shape and Tool repositories, and therefore the DSW as well, this effort has been proved to be significant in the evolution of the AIM@SHAPE network in general.

Significant restructurings have been made, mostly to the *Shape Representation* hierarchy, in the Shape Common Ontology, and to the *Shape Type* and *Functionality* hierarchies in the Tool Common Ontology. A lot of instances have been added to these hierarchies as well. This clearly strengthens the structure of the common ontologies and also increases the level of detail that is available in specifying the shapes and tools that are inserted in the respective repositories.

We have tried not to overload their structures so as not to complicate them significantly. Even so, we feel that their structure is far from being simple and future additions and modifications will run through a careful validation process, in which most partners are involved, to ensure that only subtle changes are made and large refactorings are discouraged.

4.1 Summary of changes from the previous version

The following modifications and improvements were made to the common ontologies since the previous version of this deliverable:

- Version 1.11 of the Tool Common Ontology and version 2.3 of the Shape Common Ontology have been produced.
- The common ontologies have been significantly improved to capture more domain knowledge. Progress on each of the domain ontologies can be found in the respective deliverables. The documents that constitute the latest version of these deliverables are: D1.2.1.1 – 2nd version, D1.2.2.1 – 2nd version, and D1.2.3.1 – 2nd version.
- The common ontologies are fully integrated with the domain ontologies, which means their structure is being reused by the domain ontologies and all redundancies have been eliminated.
- The common ontologies are fully integrated with the Shape and Tool repositories. The repositories now follow the structure of the common ontologies and changes in the ontologies should be immediately reflected on the repositories.