Out-of-scope predicates: projection, principles, practices, conventions, time, domain, range, class, radio, infrared, microwave, ultraviolet, xray, visible, orthogonalProjection, human, attribute, connectToNeighbors, applyTexturePt, mesh, texture, parameters, procedure, translate, graphicsOptions, lightSettings, time, date, resolution, frustum, render, rule, instance, instanceProperty, rdfLanguage, owlLanguage, flatLanguage, consistencyRequest, consistent, insertionRequest, retractionRequest, classRequest, instanceRequest, retrieveRealizedABox, otherReasonerServices, neuralNetworkEngine, otherEngine, flatData

ontologyConcept: pointConcept, segmentConcept, areaConcept, connectedSegmentsConcept, instance, class, classProperties, pointProp, elevDataProp, rasterDataProp

ontologyDataConcept: shapeType, numberOfPoints, attributeList, position, width, height, faccCode, 3DObjectPath, roll, pitch, heading

extractElevationProperties: slopeDataExtractor, watershedDataExtractor, groundLevelExtractor, elevationExtractor
extractTextureProperties: roadWidthExtractor, textureExtractor, coverageTypeExtractor, materialTypeExtractor

templateLibraryComponent: roadGen, bridgeGen, coveredBridgeGen, cantileverBeamBridgeGen, intersectionGen, walkwayGen, bicyclePathGen, trainTrackGen, rampGen, highwayGen

- Area of Interest (AOI): is a geographic region defined by rectangular limits in latitude max and min, longitude max and min in a certain coordinate system projection and model

\[ \text{aoiXY}(\text{Lat}_{\text{max}}, \text{Lat}_{\text{min}}, \text{Lon}_{\text{max}}, \text{Lon}_{\text{min}}) \equiv (\exists A)((\text{Lat}_{\text{max}}, \text{Lat}_{\text{min}}, \text{Lon}_{\text{max}}, \text{Lon}_{\text{min}}) \in A) \cap \\
(\forall P \in A)(\exists P_{\text{lat}})(\exists P_{\text{lon}})(\text{Lat}_{\text{max}} \geq P_{\text{lat}} \geq \text{Lat}_{\text{min}}) \cap (\text{Lon}_{\text{max}} \geq P_{\text{lon}} \geq \text{Lon}_{\text{min}}) \]

\[ \text{aoi}(A) \equiv (\exists A_{\text{latMax}})(\exists A_{\text{latMin}})(\exists A_{\text{lonMax}})(\exists A_{\text{lonMin}}) \text{aoiXY}(A_{\text{latMax}}, A_{\text{latMin}}, A_{\text{lonMax}}, A_{\text{lonMin}}) \]

- Culture: a principle, practice or convention within a single area and time. Applies e.g. to road signals and conventions used in a certain area, at a certain time.

\[ \text{culture}(C) \equiv (\text{principles}(C) \cup \text{practices}(C) \cup \text{conventions}(C)) \cap \text{aoi}(C) \cap \text{time}(C) \]

- Relationship: a relation between two elements belonging to the domain of discourse.

\[ \text{hasRelationshipElts}(X, Y) \equiv (\exists R)(R \in X) \cap (R \in Y) \]

\[ \text{hasRelationshipEltSet}(E, S) \equiv (\forall X \in S) \text{hasRelationshipElts}(E, X) \]

\[ \text{relatedElementsSet}(S) \equiv (\forall X \in S) \text{hasRelationshipEltSet}(X, S) \]

Unique? Between all elements.

- Domain knowledge: a set of related elements and their relationships that describe the knowledge understood in an area of study such as the field of medicine, history, a subset of cultural information, etc...

\[ \text{propertyDomain}(E, R) \equiv R \rightarrow \text{domain}(E) \]
Orthogonal imagery: imagery that is rectified and corrected from perspective skew and deformation. This is done by using multiple images of the same area such as stereo imagery.

Orthogonal Imagery $(S) \equiv (\exists O) \ orthogonalProjection(O) \cap imagery(S) \cap (\forall P \in S) \ point(P) \cap (P \in O)$
- Features: a data set representing all culture over/under or on the elevation set. This set is created using computer processes that analyze images and use pattern recognition and expert user contributions to modify and create a precise set of an area.

\[
\text{overground}(E) \equiv (\exists E_{alt}) E_{alt} \geq \text{groundAltitude}(E)
\]

\[
\text{onground}(E) \equiv (\exists E_{alt}) E_{alt} = \text{groundAltitude}(E)
\]

\[
\text{underground}(E) \equiv (\exists E_{alt}) E_{alt} \leq \text{groundAltitude}(E)
\]

\[
\text{features}(S) \equiv \left( (\forall E \in S) \text{culture}(E) \cap \left( \text{overground}(E) \cup \text{onground}(E) \cup \text{underground}(E) \right) \right)
\]

- Natural features: all features that were/are created by natural effects and circumstances such as the earth’s formations, a storm, a volcano or tsunamis such as including land, caves, tunnels and waterways.

\[
\text{humanInvolvement}(E) \equiv (\exists X) \text{human}(X) \cap (X \rightarrow E)
\]

\[
\text{naturalFeatures}(S) \equiv \text{features}(S) \cap (\forall E \in S) \neg \text{humanInvolvement}(E)
\]

- Man-made features: all features that are created or built by humans or have had some human involvement to be created. This includes the set of all urban culture as well as man-made structures like buildings and bridges and ground deformations like man-made lakes.

\[
\text{manMadeFeatures}(S) \equiv \text{features}(S) \cap (\forall E \in S) \text{humanInvolvement}(E)
\]

- Shapefiles: Restricted to representing features in terms of three basic shapes and user defined attributes per shape: points, linear and areals. A widely used and standardized format for representing features.

\[
\text{attributeList}(L) \equiv (\forall E \in L) \text{attribute}(E)
\]

\[
\text{wPoint}(E) \equiv \text{point}(E) \cap (\exists E_a) \text{attributeList}(E_a)
\]

\[
\text{wLinear}(E) \equiv (\exists E_l) \text{pointList}(E_l) \cap (\exists E_a) \text{attributeList}(E_a)
\]

\[
\text{wAreal}(E) \equiv (\exists E_l) \text{pointList}(E_l) \cap (\exists E_a) \text{attributeList}(E_a) \cap (\exists P_0 \in E_l)(\exists P_n \in E_l)(P_n = P_0)
\]

\[
\text{shapefile}(F) \equiv \text{features}(F) \cap \text{aoi}(F) \cap (\forall E \in F)(\text{wPoint}(E) \cup \text{wLinear}(E) \cup \text{wAreal}(E))
\]

- GIS Sources: also defined in Proposal report.
  - A collection of geo-correlated data sets such as elevations, imagery and features.
  - Restricted to the AOI
  - Filtered by a domain of application

\[
\text{gisSources}(S) \equiv \text{aoi}(S) \cap (\forall X \in S) \text{elevationSet}(X) \cup \text{imagery}(X) \cup \text{features}(X)
\]

- A Visualization of an AOI: represents the AOI using elevations, imagery and features to create the AOI virtually. Restricted to a certain fidelity of representation when compared to the real world. The process usually consists of:
  - Converting the elevations into a TIN. Triangulating the area using the elevation points.
- Applying the orthogonal imagery as texture on the TIN. Applying UVW texture coordinates on TIN for TIN to be correlated with imagery using the imagery and elevations respective coordinate locations and extents.
- Adding user defined complex textured 3D models, to the visualization and translating properly to the needed location on the Terrain. Here we assume the elevation set is the most accurate information. On the other hand, some more work can be done based on decision that another set has more accurate information by modifying the respective TIN for example to correct model floating issues and flatten the TIN below the models and under roads.
- Create a lighting system and an environment based on time of day and date to render the scene as close as possible to reality.

\[
\text{generateTIN}(E) \equiv \text{elevationSet}(E) \cap (\forall X \in E) \text{connectToNeighbors}(X, E)
\]

\[
\text{tin}(M) \equiv \text{mesh}(M)
\]

\[
\text{applyTexture}(M, I) \equiv \left(\exists P\right) \text{projection}(P) \cap \text{mesh}(M) \cap \text{texture}(I) \cap
(\exists I_{\text{latMax}})(\exists I_{\text{latMin}})(\exists I_{\text{lonMax}})(\exists I_{\text{lonMin}})(\exists X \in M) \text{point}(X) \cap (\exists Y \in M) \text{point}(Y) \cap
(\exists X_{\text{lat}})(\exists X_{\text{lon}})(X_{\text{lat}} = I_{\text{latMin}}) \cap (X_{\text{lon}} = I_{\text{lonMin}}) \cap (\exists Y_{\text{lat}})(\exists Y_{\text{lon}})(Y_{\text{lat}} = I_{\text{latMax}}) \cap
(\exists Y_{\text{lon}} = I_{\text{lonMax}}) \cap \text{applyTexturePt}(X, I) \cap \text{applyTexturePt}(Y, I)
\]

- A model template defined by a generation procedure and a list of attributes.

\[
\text{modelTemplate}(M) \equiv (\exists M_a)\text{attributeList}(M_a) \cap \text{procedure}(M)
\]

- a list of model templates belonging to a specific list of components

\[
\text{templateLib}(L) \equiv (\forall X \in L)\left(\text{modelTemplate}(X) \cap (\exists X_{\text{type}})(X_{\text{type}} \in \text{templateLibraryComponent})\right)
\]

- a list of model specifications: a type from the list of components and a set of parameters corresponding to the attribute list for the procedure of this component

\[
\text{modelSpec}(X) \equiv (\exists X_{\text{type}})(\exists X_i)(X_{\text{type}} \in \text{templateLibraryComponent}) \cap \text{parameters}(X_i)
\]

\[
\text{modelSpecList}(L) \equiv (\forall X \in L)\text{modelSpec}(X)
\]

- a visual model: (geometry and texture) or (a template and parameters)

\[
\text{visModelInstance}(M) \equiv
(\exists M_a)(\exists M_i)(\text{mesh}(M_a) \cap \text{texture}(M_i) \cup (\exists M_a)(\exists M_i)\text{modelTemplate}(M_i) \cap \text{parameters}(M_i) \cap (\forall X \in M_i) (X \in M_i))
\]

\[
\text{visModelList}(L) \equiv (\forall X \in L)\text{visModelInstance}(X)
\]

\[
\text{terrain}(T) \equiv (\exists M)\text{tin}(M) \cap (\exists I)\text{orthogonalImagery}(I) \cap \text{texture}(I)
\]

- places instance on terrain. Can be modified to take another sources as ground truth and modify terrain and other sets appropriately.

\[
\text{placeInstance}(X, T) \equiv (\exists P \in T)(\exists P_{\text{alt}})(P_{\text{alt}} = \text{groundAltitude}(X)) \cap \text{translate}(X, P)
\]
- generates visual model from a model specification using a template library and components

\[ \text{generateVisModel}(X, L) \equiv \text{templateLib}(L) \cap (\exists X_{\text{type}})(\exists X_i)(\exists X_{\text{inst}})(\exists Y \in L) \left( (\exists Y_{\text{type}} = X_{\text{type}}) \cap \text{modelTemplate}(Y) \cap \text{parameters}(X_i) \rightarrow X_{\text{inst}} \right) \cap \text{visModelInstance}(X_{\text{inst}}) \]

\[ \text{addModels}(T, S, L) \equiv \text{terrain}(T) \cap \text{modelSpecList}(S) \cap \text{templateLib}(L) \cap (\forall X \in S) \text{ generateVisModel}(X, L) \cap \text{placeInstance}(X, T) \]

\[ \text{renderSettings}(X) \equiv \text{aoi}(X) \cap \text{graphicsOptions}(X) \cap \text{lightSettings}(X) \cap \text{time}(X) \cap \text{date}(X) \cap \text{resolution}(X) \cap \text{frustum}(X) \]

\[ \text{aoiProceduralVisualization}(E, I, S, L) \equiv \text{elevationSet}(E) \cap \text{imagery}(I) \cap (\exists S_i) \text{ modelSpecList}(S_i) \cap (\exists S_i) \text{ visModelList}(S_i) \cap \text{templateLib}(L) \cap \text{aoi}(\{E \cup I \cup S\}) \cap (\exists N) (\exists T) (\exists M) (\exists R) \left( \text{generateTIN}(E) \rightarrow N \cap \text{tin}(N) \right) \cap \left( \text{applyTexture}(N, I) \rightarrow T \cap \text{terrain}(T) \right) \cap \left( \text{addModels}(T, S, L) \cap (\forall X \in S_i) \text{ placeInstance}(X, T) \rightarrow M \cap \text{visModelList}(M) \right) \cap \text{renderSettings}(R) \cap \text{render}(T, M, R) \]

- TBox: the Terminology box. The set of all domain concepts and their relationships. A formal ontology of terms and their logical associations to one another. Many standards in representing TBoxes are available. The most widely used is the OWL 2.0 representation. It is used as input to a reasoner. In our system, the TBox defines transportation related axioms and the transportation’s elements relationships together in North America. For example, a railroad crossing a street defines railroad signals and blocking gates that activate when the train is at a certain distance. Another example is: there is an intersection where two main streets cross each other and there are traffic lights at this intersection that model intersection lights.

\[ \text{tBox}(S) \equiv (\exists C)(\exists R) \text{ domainKnowledge}(C, R) \cap \left( (\forall X \in S) (X \in C) \cup (X \in R) \cup \text{rule}(X) \right) \]

- ABox: the Assertion box. A formal ontology of instances of terms defined by the TBox and the instances relationships. Also widely used through the OWL 2.0 standard. Both TBox and ABox can be coexistent in the same ontology. The ABox can be created dynamically and some reasoners offer retraction services to remove a fact from the ABox and resolve.

\[ \text{aBox}(S) \equiv (\exists C)(\exists R) \text{ domainKnowledge}(C, R) \cap (\forall X \in S) \text{ instance}(X) \cap (\exists X_c) \text{ class}(X_c) \cap (X_c \in C) \cap \left( (\exists X_p) \text{ instanceProperty}(X_p) \rightarrow (X_p \in R) \right) \]

- Knowledge Base: the available knowledge, the union of the TBox and the ABox.

\[ \text{knowledgeBase}(K) \equiv (\exists K_i) \text{ tBox}(K_i) \cap (\exists K_a) \text{ aBox}(K_a) \]

- Semantic Web reasoner: is a description logic compatible reasoner that operates on the TBox and ABox to service search and information request queries from client applications. Reasoners are widely researched and their capabilities are being enhanced. But there are only a certain class that are standardized and available to work with. Different reasoners can accept different expressivity for reasoning and the more evolved reasoners are generally based on Tableaux algorithms and calculus and support expressive description logics including all of OWL DL (SHOIN(D)) constructs with some restrictions. Most reasoners support OWL Lite, and some support OWL FULL, OWL DL lies
in the middle. The SHOIN expressivity has the advantage of being decidable even for consistency checking. The newer expressivity algorithms such as SHOIQ (extends SHOIN with Qualified number restrictions) and SROIQ (extends SHOIN with complex role inclusion axioms) support. We are basing our research mainly on SROIQ(D) which has optimal expressivity to include role disjointness and supports datavalues or datatypes reasoning. Only Pellet and Fact++ support this expressivity. We chose Pellet due to SWRL rule support capability. OWL2 supports SROIQ(D) but the Protégé editor is limited to SHOIN(D) (OWL-DL).

\[
\text{expressivity}(K) \equiv (\forall X \in K) \ X \in \{ S, R, H, O, I, N, Q, (D) \}
\]

\[
\text{swLanguage}(L) \equiv \text{rdfLanguage}(L) \cup \text{owlLanguage}(L) \cup \text{flatLanguage}(L)
\]

\[
\text{swKnowledgeBase}(K) \equiv \text{knowledgeBase}(K) \cap \text{expressivity}(K) \cap \text{swLanguage}(K)
\]

\[
\text{consistencyQuery}(K, Q) \equiv \text{consistencyRequest}(Q) \to \text{consistent}(K)
\]

\[
\text{insertionQuery}(K, Q) \equiv (\text{insertionRequest}(Q) \cap \text{consistent}(K)) \to (\exists K_2 \text{swKnowledgeBase}(K_2) \cap (K \in K_2) \cap (Q \in K_2) \cap \text{consistent}(K_2))
\]

\[
\text{retractionQuery}(K, Q) \equiv (\text{retractionRequest}(Q) \cap \text{consistent}(K)) \to (\exists K_2 \text{swKnowledgeBase}(K_2) \cap (K \in K) \cap (Q \notin K_2) \cap \text{consistent}(K_2))
\]

\[
\text{classQuery}(K, Q) \equiv (\text{classRequest}(Q) \cap (\exists X \in K) (\exists Q_c) \text{class}(X) \cap \text{class}(Q_c) \cap (X = Q_c)) \to X
\]

\[
\text{instanceQuery}(K, Q) \equiv (\text{instanceRequest}(Q) \cap (\exists X \in K) (\exists Q_i) \text{instance}(X) \cap \text{instance}(Q_i) \cap (X = Q_i)) \to X
\]

\[
\text{swReasoner}(K, Q) \equiv \text{swKnowledgeBase}(K) \cap (\text{consistencyQuery}(K, Q) \cup \text{insertionQuery}(K, Q) \cup \text{retractionQuery}(K, Q) \cup \text{classQuery}(K, Q) \cup \text{instanceQuery}(K, Q) \cup \text{retrieveRealizedABox}(K, Q) \cup \text{otherReasonerServices}(K, Q))
\]

- Knowledge base system: a system composed of a service engine and data that the engine uses to service queries from applications. Semantic web is an example of a knowledge base system where the data is in form of formal static TBox and dynamic ABox and a semantic web reasoner or engine.

\[
\text{serviceEngine}(E) \equiv \text{swReasoner}(E) \cup \text{neuralNetworkEngine}(E) \cup \text{otherEngine}(E)
\]

\[
\text{engineData}(K) \equiv \text{swKnowledgeBase}(K) \cup (\exists K_c) (\exists K_r) \text{domainKnowledge}(K_c, K_r) \cup \text{flatData}(K)
\]

\[
\text{kbSystem}(E, K) \equiv \text{serviceEngine}(E) \cap \text{engineData}(K)
\]

- GIS2KB: our process that converts GIS sources information into knowledge for the knowledge base system to use. A methodology for this process has been defined. Its inputs are elevations, imagery and Shapefiles of a certain area and a TBox specific to the domain. Two pieces constitute the TBox, a data TBox and a domain TBox. The data TBox converts data specific from the types in the sources to the concepts in the domain (a bridge ontology). The domain TBox implements the domain knowledge which will allow semantic reasoning about the assertions inserted. Our technique is to interpret and transform the data available in the GIS sources into assertions in the knowledge base usable by a semantic web knowledge base system using OWL 2. Due to its advantages when reasoning on qualified cardinality restrictions and support for complex role inclusion axioms and datavalues and datatype support reasoning, SROIQ(D) expressivity will be used and we will evaluate...
its main advantages and disadvantages when compared to SHOIQ and SHIQ reasoners. We will also express the advantages of the P-SROIQ which is in the works adding the capability of probabilistic axioms to SROIQ. We extract available information from the GIS sources corresponding to the data ontology being used. In this case we are only interested in the transportation features. All other features are left as they are described and rendered correspondingly.

domainConcept(C) \equiv (\exists D) \text{ domain}(D) \land (C \in D) \land \text{ class}(C)

dataConcept(C) \equiv (\exists D) \text{ gisSources}(D) \land (\exists X \in D) \text{ class}(X) \land (C = X)

domainTBox(T) \equiv
tBox(T) \land (\forall P) \text{ relationshipProperty}(P, T) \land \text{ propertyDomain}(E_1, P) \land 
domainConcept(E_1) \land \text{ propertyRange}(E_2, P) \land \domainConcept(E_2)
dataTBox(T) \equiv tBox(T) \land (\forall P) \text{ relationshipProperty}(P, T) \land \text{ propertyDomain}(E_1, P) \land 
dataConcept(E_1) \land \text{ propertyRange}(E_2, P) \land \domainConcept(E_2)

protegeTransportationOWLOntology(T) \equiv \text{ domainTBox}(T) \land (\forall X \in T) \left( \text{ class}(X) \rightarrow \ontologyConcept(X) \right)

shapefileTransportationConceptsMapping(T) \equiv
dataTBox(T) \land (\exists C) \text{ protegeTransportationOWL Ontology}(C) \land 
(\forall X \in T) \left( \text{ domainConcept}(X) \rightarrow (X \in C) \right) \land 
(\forall Y \in T) \left( \text{ dataConcept}(Y) \rightarrow \ontologyDataConcept(Y) \right)

addToKB(P, K) \equiv \text{ swReasoner}(K, P) \land \text{ insertionQuery}(K, P)

map(I, A, C) \equiv (\forall X \in C) (\exists I_x)(\exists A_x)(I_x = A_x)

- For all Shapefile sources, the data is converted into equivalent data using the expressivity mentioned above according to the defined data ontology that will instance domain concepts. These will be processed by the KB2Vis process as specified above to add on-ground features to the visualization. More is defined in the GIS2KB report.

\* (non 1-1) \*

mapPropertiesToInstance(I, A, T) \equiv \text{ instance}(I) \land \text{ attributeList}(A) \land \text{ dataTBox}(T) \land 
(\exists Q) \text{ classQuery}(T, Q) \land (I \in Q) \land \text{ swReasoner}(T, Q) \rightarrow (\exists X \in T) \text{ class}(X) \land \text{ map}(I, A, X)

mapPoint(E, T, K) \equiv

wPoint(E) \land \text{ dataTBox}(T) \land \text{ swKnowledgeBase}(K) \land (\exists P) \text{ instance}(P) \land 
(\exists P_c) \text{ pointConcept}(P_c) \land \text{ map}(P, E, P_c) \land (\exists E_a) \text{ attributeList}(E_a) \land 
\text{ mapPropertiesToInstance}(P, E_a, T) \land \text{ addToKB}(P, K)

createSegment(S, P_1, P_2) \equiv \text{ segmentConcept}(S) \land (\exists X \in P_1) \text{ pointConcept}(X) \land 
\text{ map}(S, P_1, X) \land (\exists Y \in P_2) \text{ pointConcept}(Y) \land \text{ map}(S, P_2, Y)

createProperty(P, S_1, S_2) \equiv \text{ instanceProperty}(P) \land \text{ hasFormalRelationship}(P, S_1, S_2)

mapLinear(E, T, K) \equiv

wLinear(E) \land \text{ dataTBox}(T) \land \text{ swKnowledgeBase}(K) \land 
(\exists E_1)(\forall E_{i0})(\exists E_{i1})(\exists E_{i2})(\exists S_1) \text{ instance}(S_1) \land \text{ createSegment}(S_1, E_{i0}, E_{i1}) \land 
(\exists E_a) \text{ attributeList}(E_a) \land \text{ mapPropertiesToInstance}(S_1, E_a, T) \land (\exists S_2) \text{ instance}(S_2) \land 
\text{ createSegment}(S_2, E_{i1}, E_{i2}) \land \text{ mapPropertiesToInstance}(S_2, E_a, T) \land 
(\exists P) \text{ instanceProperty}(P) \land \text{ connectedSegmentsConcept}(P) \land 
\text{ createProperty}(P, S_1, S_2) \land \text{ addToKB}(S_1, K) \land \text{ addToKB}(S_2, K) \land \text{ addToKB}(P, K)
setPerimeterPoint\((A, P)\) ≡
(∃R ∈ A) pointProp\(R\) \cap propertyRange\(P, R\) \cap propertyDomain\(A, R\)

mapAreal\((E, T, K)\) ≡
wAreal\(E\) \cap dataTBox\(T\) \cap swKnowledgeBase\(K\) \cap (∃A) instance\(A\) \cap
(∃A_c) areaConcept\(A_c\) \cap map\(A, E, A_c\) \cap (∃E_t) pointList\(E_t\) \cap
(∀P ∈ E_t) setPerimeterPoint\((A, P)\) \cap (∃E_a) mapPropertiesToInstance\(A, E_a, T\) \cap
addToKB\(A, K\)

mapShapefile\((S, T, K)\) ≡
shapefile\(S\) \cap shapefileTransportationConceptsMapping\(T\) \cap
(∀X ∈ S)\((wPoint\(X\) → mapPoint\(X, T, K\)) \cap (wLinear\(X\) → mapLinear\(X, T, K\)) \cap
(wAreal\(X\) → mapAreal\(X, T, K\))

(1-1)

mapShapefile\((S, T, K)\) ≡
shapefile\(S\) \cap ShapefileTransportationConceptsMapping\(T\) \cap
(∀X ∈ S)\((∃Q) classRequest\(Q\) \cap (∃X_c) class\(X_c\) \cap \(X_c \in Q\) \cap swReasoner\(T, Q\) →
(∃C)(class\(C\) \cap \(C \in T\)) \cap (∃I) instance\(I\) \cap map\(I, X, C\) \cap addToKB\(I, K\)

- If elevation files are defined, terrain class instance(s) are defined in the knowledge base and they
  have a pointer to their corresponding datasets. The KB2Vis process gets the terrain instances if they
  are defined and are in the range being visualized and renders a polygonal mesh of the elevations
  available. More is defined in GIS2KB report.

- If imagery is available, imagery class instance(s) are defined in the knowledge base and they have
  a pointer to their corresponding datasets. The KB2vis process gets uses the corresponding image to
  texture the created polygonal mesh. More is defined in the GIS2KB report. For each Shapefile
  feature, information is extracted from the imagery to create a texture pointer for the feature’s
  visualization. Elevation is evaluated and identifiable properties from the imagery are added as
  knowledge to the knowledge base. More is defined in the GIS2KB report.

hasProperty\((E, P)\) ≡ instance\(E\) \cap (∃K) swKnowledgeBase\(K\) \cap ontologyConcept\(P\) \cap
relationshipProperty\(P, K\) \cap (∃Q) classRequest\(Q\) \cap (∃E_c) class\(E_c\) \cap \(E_c \in Q\) \cap
swReasoner\(K, Q\) → (∃C) class\(C\) \cap \(P \in C\) \cap propertyDomain\(E, P\)

inArea\((E, S)\) ≡ aoi\(S\) \cap (∃E_{lat})(∃E_{lon})(∃S_{latMin})(∃S_{latMax})(∃S_{lonMin})(∃S_{lonMax})(S_{latMax} ≥
\(E_{lat} ≥ S_{latMin}\) \cap \(S_{lonMax} ≥ E_{lon} ≥ S_{lonMin}\)

extractElevations\((S, A, E)\) ≡ aoi\(E\) \cap (∀P)(P \in A) \cap point\(P\) \cap (∃P_{elev})\(P_{elev} =
groundAltitude\(P\)) \cap \(P \in S\)

mapElevations\((E, K)\) ≡
elevationSet\(E\) \cap (∀X)\((X \in K) \cap instance\(X\) \cap (∃P) elevDataProp\(P\) \cap
hasProperty\(X, P\) \cap inArea\(X, E\) →
(∃S) elevationSet\(S\) \cap extractElevations\((S, X, E)\) \cap propertyRange\(S, P\) \cap
extractElevationProperties\(X\) \cap consistent\(K\)
extractTexture(S, A, I) ≡
  aoi(I) ∩ (∀P)(P ∈ A) ∩ point(P) ∩ (∃P_{lat})(∃P_{lon})(∃I_{px})(∃I_{py})(P_{lat} = I_{px}) ∩ (P_{lon} = I_{py}) ∩ (∃P_v)(P_v = I_{pv}) ∩ (P ∈ S)
mapImagery(I, K) ≡
  imagery(I) ∩ (∀X)(X ∈ K) ∩ instance(X) ∩ (∃P) rasterDataProp(P) ∩ hasProperty(X, P) ∩ inArea(X, I) → (∃S) texture(S) ∩ extractTexture(S, X, I) ∩ propertyRange(S, P) ∩ extractTextureProperties(X) ∩ consistent(K)
GIS2KB(E, I, S, T) ≡
  shapefileTransportationConceptsMapping(T) ∩ (∃K)swKnowledgeBase(K) ∩ (∀X ∈ S) shapefile(X) ∩ mapShapefile(X, T, K) ∩ (∀Y ∈ E) elevationSet(Y) ∩ mapElevations(Y, K) ∩ (∀Z ∈ I) imagery(Z) ∩ mapImagery(Z, K)

--
resolutionSort(R) ≡ (∀X ∈ R)(∃P ∈ X) layerResolution(P) ∩ insert(X, R[P])
assignDefault(P, K) ≡ (∃Q) classRequest(Q) ∩ swReasoner(K, Q)
  → (∃C) class(C) ∩ defaultValue(C_v) ∩ (P = C_v)
runExtractor(E, S, O) ≡ (∀L ∈ S) layer(L) ∩ (getNextLayer(L, S) ∩ extract(E, I, O))
  → (∃E_p ∈ E) propertyValues(E_p) ∩ valid(E_p)
mapProperties(O, E, I, K)
  ≡ (∃M ∈ P) extractorDefinition(M)
  ∩ ((imageryType(M_{type}) → runExtractor(M, I, O))
  ∩ (elevationType(M_{type}) → runExtractor(M, E, O))
  ∩ (shapeType(M_{type}) → extract(M, K, O) ∩ (∃M_v ∈ M) propertyValues(M_v))
  ∩ (valid(M_v) → (P = M_p)) ∪ assignDefault(P)
addToKB(P, K) ≡ swReasoner(K, P) ∩ insertionQuery(K, P)
map(I, A, C) ≡ (∀X ∈ C)(∃I_x)(∃A_x)(I_x = A_x)
mapShapefile(S, T, K)
  ≡ shapefile(S) ∩ shapefileTransportationConceptsMapping(T)
  ∩ (∀X ∈ S)(∃Q) classRequest(Q) ∩ (∃X_c) class(X_c) ∩ (X_c ∈ Q) ∩ swReasoner(T, Q)
  → (∃C)(class(C) ∩ (C ∈ T)) ∩ (∃I) instance(I) ∩ map(I, X, C) ∩ addToKB(I, K)
GIS2KB(E, I, S, T)
  ≡ shapefileTransportationConceptsMapping(T) ∩ (∃K)swKnowledgeBase(K)
  ∩ (∀X ∈ S) shapefile(X) ∩ mapShapefile(X, T, K) ∩ consistent(K)
  ∩ resolutionSort(E) ∩ resolutionSort(I) ∩ (∀O ∈ K) instance(O)
  ∩ (∀P ∈ O) invalid(P) → (mapProperties(O, E, I, K) ∩ consistent(K))

- KB2Vis: our process that converts the knowledge in the knowledge base system (implicit knowledge made explicit by the engine) into visualization input data for the visualization system. This process uses SPARQL for the querying of the knowledge base and the retrieval of the instances needed to
form the visualization. The input is a Visualization capabilities ontology, denoted as the 3D Parametrized Model Library, an area to visualize (bounding box) determined by the viewpoint needed and settings on what type of information and the depth. The process retrieves all instances in the corresponding area from the knowledge base and depending on the visualization system’s capabilities (Visualization capabilities ontology), the system retrieves available data values and explicit information from the knowledge base to feed its internal procedural graphics functions with the available data and generate the visualization.

\[
createModelSpec(M, I, K, L) \\
\equiv (\exists M_t) parameters(M_t) \cap (\exists I_{type})(\exists I_{type})(M_{type} = I_{type}) \\
\land (\exists T \in L)(\exists T_{type} = I_{type})(\exists T_t) attributeList(T_t) \rightarrow map(M_t, I, T_t)
\]

\[
existing3DObjectRef(Y, X) \\
\equiv (\exists P) hasProperty(X, P) \cap 3DObjectPath(P) \cap propertyRange(Y, P) \\
\land visModelInstance(Y)
\]

\[
KB2VIS(S, K, L) \\
\equiv (\exists S_i) modelSpecList(S_i) \cap (\exists S_i) visModelList(S_i) \cap (S \rightarrow S_i \cup S_i) \\
\land swKnowledgeBase(K) \cap (L \in K) \land templateLib(L) \\
\land (\exists Q) retrieveRealizedabox(Q) \cap swReasoner(K, Q) \\
\rightarrow ((\exists R) aBox(R) \land (\forall X \in R) instance(X) \\
\land (\exists Y \in S)(createModelSpec(Y, X, K, L) \cup existing3DObjectRef(Y, X)))
\]

- Main process uses the above components to describe the full system. The process fuses information from the different datasets together and handles ambiguity and inconsistencies in the data through the semantic web system and visualizes the inferred data appropriately.

\[
mainProcess(E, I, F) \equiv \\
elevationSet(E) \cap imagery(I) \cap shapefile(F) \cap (\exists K) swKnowledgeBase(K) \land \\
(\exists L \in K) templateLib(L) \land (\exists T \in K) shapefileTransportationConceptsMapping(T) \land \\
GIS2KB(E, I, F, T) \land (\exists S)(3S_i) modelSpecList(S_i) \cap (\exists S_i) visModelList(S_i) \land \\
(S \rightarrow S_i \cup S_i) \land consistent(K) \land KB2VIS(S, K, L) \rightarrow \\
aoiProceduralVisualization(E, I, S, L)
\]
shape elt1 as new instance(I)

A: FACC 3 = sec. road (I)
  FACC 5 = major road
  width = road width (80%) and walk width (20%)

B: sec. road = transport 0...n...
  major road =
  road width = 6 transport
  walkway width =
  bridge =

+ many data elements + inferencing

C: template
  template model attributes
  instance (I) -> bridge procedure
  bridge query (road width...