Formalization of 2-D Spatial Ontology and OWL/Protégé Realization

Kulsawasd Jitkajornwanich
Computer Science & Engineering Department
University of Texas at Arlington
kulsawasd.jitkajornwanich@mavs.uta.edu

Ramez Elmasri
Computer Science & Engineering Department
University of Texas at Arlington
elmasri@cse.uta.edu

Chengkai Li
Computer Science & Engineering Department
University of Texas at Arlington
cli@cse.uta.edu

John McEnery
Department of Civil Engineering
University of Texas at Arlington
mcenery@uta.edu

ABSTRACT
Ontology specification is a core component of the Semantic Web, and facilitates interoperability among different systems that use distinct models. Developing a spatial ontology will allow many applications that have spatial objects to interact. In this paper, we formalize 2-D spatial concepts and operations into a spatial ontology. We show how these concepts can be realized in Protégé. The Protégé spatial ontology provides spatial built-ins that can be used to provide a spatial dimension to other ontologies when needed. We give some examples of the use of our ontology, which is based on a standard Geometry class hierarchy [Consortium 1999], with a few modifications.

Keywords
Ontology, Spatial, Semantic web, Protégé, Built-ins, Lightweight, Formalization

1. INTRODUCTION
Ontology [Gruber 2008; Noy and Deborah 2000] is considered to be a core component of the Semantic Web [Bechhofer et al 2003]. With the reasoning, inference, and representation mechanisms associated with an ontology, it becomes possible that systems with different definitions of the same concepts can interoperate with each other. In addition, a nearly complete description of concepts in a particular area of knowledge becomes readily available for interest ed users. In this paper we focus on spatial ontology. Representing spatial knowledge is a basic problem in many applications, such as GIS and map applications. In the past few years, work on spatial ontologies has focused on two main areas: spatial database integration [Bennacer et al 2003; Bittner et al 2006; Hess et al 2006] and spatial ontology creation [Baglioni et al 2007; Spaccapietra et al 2004]. In spatial database integration, a spatial ontology is used as a tool to integrate different spatial databases. In spatial ontology creation, there are two different major approaches. First, by analyzing a collection of existing spatial databases and methodologies, a spatial ontology model is defined based on those databases [Baglioni et al 2007]. However, this leads to the problem that the created spatial ontology will be limited to those databases and consequently will not be sufficient to be a standard for representing a complete formal spatial ontology. The second approach in spatial ontology creation is to define a complete spatial ontology model. For example, in [Spaccapietra et al 2004], they propose to extend MADS, an approach that allows a regular database to model spatial and temporal characteristics, into their previous work to capture spatial ontology [Parent et al 1999, 2006]. However, this approach has not been materialized in a system, and hence is difficult to utilize. There is no formal specification of spatial ontology developed from this approach. In addition, it is limited to the polygon data type only. Thus, the complete set of operations among point, line, and polygon is lacking.

Our work gives a formal specification of spatial ontology, defines a complete collection of spatial operations, and provides a general spatial ontology implemented on Protégé. Many researchers have been working in the area of Temporal and Spatial Ontology [Milea et al 2009; Spaccapietra et al 2004; Bennacer et al 2004; Baglioni et al 2007; Bittner et al 2006; Mark et al 2000; O'Connor and Amar 2010; Hobbs and Feng 2004; Baratis et al 2009; Hess et al 2006; Abdelmoty et al 2009; Andronikos et al 2009]. A specification of temporal ontology was introduced in [Hobbs and Feng 2004]. It clearly discussed temporal ontology formalization, and comprehensively covers the definitions of temporal concepts and operations. It is based on the temporal logic developed by Allen [Allen 1983; Allen and Kautz 1985]. The following is an example of the Meet operation between two time intervals formalized by the paper, assuming that $T_1$ and $T_2$ are two time intervals.

$$\text{Meet}(T_1,T_2) = (\exists t)[\text{ends}(t,T_1) \land \text{begins}(t,T_2)]$$

A complete formalization of ontology forms the basis and reference for ontology implementation. In addition, since the temporal ontology specification in [Hobbs and Feng 2004] was intended to capture all temporal reasoning on the web pages, it is gradually becoming the standard for temporal ontology specification.

One of the main applications of spatial ontology is GIS applications. Although spatial concepts and operations have been specified in many works [Consortium 1999; Parent et al 1999; Shekhar and Sanjay 2003; Wang et al 2000; Parent et al 2006], there are few attempts at specifying a complete formal ontology for spatial concepts. Spatial operations are more complex than temporal operations, and can be defined over multiple dimensions, especially two and three dimensions, whereas temporal operations are only on one dimension. Figure 1 shows how the Meet operation is different in one dimension and two dimensions. Additionally, temporal operations have only two directions (before and after) whereas for two dimensional spatial operations, there are continuous directions along 360º of a two dimensional space. Figure 2 shows eight directions, at 45º intervals. (East is 0º, north is 90º, west is 180º, etc.).
The rest of the paper is organized as follows. Section 2 discusses the spatial ontology formalization consisting of spatial concept definitions and spatial operation definitions. Section 3 discusses how the defined ontology is implemented in Protégé, along with some small examples to show how it works. Section 4 discusses related work. Section 5 presents conclusion and future work.

2. SPATIAL ONTOLOGY FORMALIZATION

2.1 Formalization of Concept Definitions

In [Open GIS Consortium 1999], a geometry class hierarchy is proposed for 2D objects. The hierarchy shown in Figure 4 is based on the one in [Open GIS Consortium 1999], with some minor modifications to allow our formalization.

![Figure 4. Geometry class hierarchy](image)

Considering the leaf nodes in the hierarchy of Figure 4, the geometry objects can be categorized into 8 types as shown in Figure 5.

1. Point (p)
2. Single Line (sl)
3. Connected Line (cl): Non-Ring (nr)
4. Connected Line (cl): Ring (r)
5. Polygon (a)
6. MultiPoint (mp)
7. MultiCurve (mc)
8. MultiPolygon (ma)

![Figure 5. Types of geometry object in 2-D space](image)

Our proposed spatial ontology consists of two parts: concept definitions and operation definitions. The following is the formalization of concept definitions.

In order to be used in the ontology, the concept definitions of all geometry object types in two-dimensional space (see Figure 4 and 5) have to be defined. For simplicity, we shall assume the 2D coordinate system based on longitude and latitude, although the formalization can be adapted to other 2D coordinate systems.
1. **Point** \((p)\)

Point can be defined by longitude \(x\) and latitude \(y\).

\[\text{Point}(p) \equiv (x, y) \text{ where } x = \text{longitude} \text{ and } y = \text{latitude} \]

2. **Single Line** \((sl)\)

Single line can be defined by any two points.

\[\text{SingleLine}(sl) \equiv (p_1, p_2) \text{ where } p_1 \neq p_2 \]

Single line also includes some unary operations, such as \(\text{slope}(m, sl)\) and \(\text{distance}(d, sl)\), which are defined in Section 2.2.2.

3. **Connected Line** \((cl)\): Non-Ring \((nr)\)

Non-ring connected line can be defined by a sequence of points \((p_1, p_2, ..., p_N)\), \(N \geq 3\), and \(p_i \neq p_j\) for \(i \neq j\), which in turn defines a sequence of single lines \((s_l_1, s_l_2, ..., s_l_{N-1})\) and each \(s_l_i = (p_{i+1} - p_i)\) for \(i = 1, 2, ..., N-1\) and \(s_l_N = (p_1 - p_N)\). However, the area inside the ring is not part of \((sl)\). The valid connected line is defined as follows:

\[\text{RingConnectedLine}(cl; nr) \equiv (p_i | 1 \leq i \leq N, N \geq 3, \text{ and } p_i \neq p_j \text{ for } i \neq j, \text{ which in turn defines a sequence of single lines } (s_l_1, s_l_2, ..., s_l_{N-1}) \text{ and each } s_l_i = (p_{i+1} - p_i) \text{ for } i = 1, 2, ..., N-1 \text{ and } s_l_N = (p_1 - p_N), \text{ and the area within is part of the polygon.} \]

\[\text{Polygon}(a) \equiv (p_i | 1 \leq i \leq N, N \geq 3, p_i \neq p_j \text{ for } i \neq j, \text{ which in turn defines a sequence of single lines } (s_l_1, s_l_2, ..., s_l_{N-1}) \text{ and each } s_l_i = (p_{i+1} - p_i) \text{ for } i = 1, 2, ..., N-1 \text{ and } s_l_N = (p_1 - p_N), \text{ and the area within is part of the polygon.} \]

4. **Point and Line**

Point and line can be defined by any two points.

\[\text{PointLine}(pl) \equiv (p, sl) \text{ where } p \neq sl \]

Point and line also includes some unary operations, such as \(\text{slope}(m, pl)\) and \(\text{distance}(d, pl)\), which are defined in Section 2.2.2.

5. **Point and Single Line**

Point and single line can be defined by any two points.

\[\text{PointLine}(pl) \equiv (p, sl) \text{ where } p \neq sl \]

Point and single line also includes some unary operations, such as \(\text{slope}(m, pl)\) and \(\text{distance}(d, pl)\), which are defined in Section 2.2.2.

6. **Point and Connected Line**

Point and connected line can be defined by any two points.

\[\text{PointConnectedLine}(pc) \equiv (p, cl) \text{ where } p \neq cl \]

Point and connected line also includes some unary operations, such as \(\text{slope}(m, pc)\) and \(\text{distance}(d, pc)\), which are defined in Section 2.2.2.

7. **End Point**

End point is a relationship between point and single line. In the formalization, we sometimes need to distinguish unambiguously the two endpoints so we define two operations on a line: \(\text{EndPointNe} \text{ and EndPointSw}\). North and south will have precedence in distinguishing the endpoints and east and west will be used only if a line is horizontal. That is, the end point of the line with higher latitude will be classified as \(\text{EndPointNe}\) regardless of whether the longitude is either east or west. Figure 6(a) shows some examples of how the endpoints are categorized.

A point \(p\) will be an end point of single line \(sl\) if and only if \(p\) is equal to either one of the single line endpoints.

\[\text{EndPoint}(p, sl) \equiv p = sl.p_1 \lor p = sl.p_2 \]

\[\text{EndPointNe}(p, sl) \equiv \text{EndPoint}(p, sl) \land (\exists p_1)[\text{EndPoint}(p_1, sl) \land p 
\neq p_1 \land [(sl.p \cdot y \geq sl.p_1.y) \lor (sl.p \cdot y = sl.p_1.y \land sl.p \cdot x > sl.p_1.x)]]\]

\[\text{EndPointSw}(p, sl) \equiv \text{EndPoint}(p, sl) \land (\exists p_1)[\text{EndPoint}(p_1, sl) \land p 
\neq p_1 \land [(sl.p \cdot y < sl.p_1.y) \lor (sl.p \cdot y = sl.p_1.y \land sl.p \cdot x < sl.p_1.x)]]\]

The relationship between \(\text{EndPointNe}\) and \(\text{EndPointSw}\) can be specified as follows:

\[\text{EndPointNe}(p_1, sl) \land \text{EndPointSw}(p_2, sl) \Rightarrow [sl.p_1.y_1 > sl.p_2.y_2 \lor (sl.p_1.y_1 = sl.p_2.y_2 \land sl.p_1.x_1 > sl.p_2.x_2)]\]

We can now define the two unary operations on single line, \(\text{slope}\) and \(\text{distance}\), as follows:

\[\text{Slope}(m, sl) \equiv \text{EndPointNe}(p_1, sl) \land \text{EndPointSw}(p_2, sl) \land m = [(p_1.y - p_2.y)/(p_1.x - p_2.x)]\]

\[\text{Distance}(d, sl) \equiv \text{EndPointNe}(p_1, sl) \land \text{EndPointSw}(p_2, sl) \land d = \sqrt{((p_1.y - p_2.y)^2 + (p_1.x - p_2.x)^2)}\]

8. **Oblique Line**

Oblique line is a relationship between point and single line. If a point \(p\) is on the single line \(sl\) then the slope of the point \(p\) to one of the endpoints of the single line must be equal to the slope of the single line. In addition, the point \(p\) has
to fall on the single line.

\[ \text{Ontheline}(p, sl) \equiv \text{Slope}(m_1, sl) = \text{Slope}(m_2, sl) \lor \text{Slope}(m_2, p) = \text{Slope}(m_1, p) \land m_1 = m_2 \land (sl.p_1.x_1 \leq p.x \leq sl.p_2.x_2 \lor sl.p_2.x_2 \leq p.x \leq sl.p_1.x_1) \]

An endpoint is also considered to be on the line.

\[ \text{Endpoint}(p, sl) \rightarrow \text{Ontheline}(p, sl) \]

2.2.3 Single line and Single line

2. There are five main relationships between single lines.

1. Equal

Two single lines are equal if and only if they have exactly the same points.

\[ \text{Equal}(sl_1, sl_2) \equiv sl_1.p_1 = sl_2.p_1 \land sl_1.p_2 = sl_2.p_2 \]

2. Meet

Two single lines meet when they share one endpoint. We can further divide the operation into two more cases: MeetSameSlope and MeetDiffSlope as follows (see Figure 6(b,c)).

\[ \text{Meet}(sl_1, sl_2) \equiv (\exists p \in \{sl_1.p_1, sl_1.p_2, sl_2.p_1, sl_2.p_2\}) \land \text{Endpoint}(p, sl_1) \land \text{Endpoint}(p, sl_2) \]

\[ \text{MeetSameSlope}(sl_1, sl_2) \equiv \text{Meet}(sl_1, sl_2) \land \text{Slope}(m_1, sl_1) = \text{Slope}(m_2, sl_2) \land m_1 = m_2 \]

\[ \text{MeetDiffSlope}(sl_1, sl_2) \equiv \text{Meet}(sl_1, sl_2) \land \text{Slope}(m_1, sl_1) \neq \text{Slope}(m_2, sl_2) \land m_1 \neq m_2 \]

3. Cross

Two single lines cross when their slopes are different and the intersecting point falls on both lines. To make formalization easier, we will first define Between and ProperBetween relation as follows (see Figure 6(e)). We also have an operation \( \text{IntersectingPoint}(p, sl_1, sl_2) \) that returns the point of intersection \( p \) between two single lines \( sl_1, sl_2 \) if the two lines do not have the same slope. We do not show this because of limited space.

\[ \text{Between}(x, x_1, x_2) \equiv (x_1 \leq x \leq x_2 \lor x_2 \leq x \leq x_1) \]

\[ \text{ProperBetween}(x, x_1, x_2) \equiv (x_1 < x < x_2 \lor x_2 < x < x_1) \]

\[ \text{Cross}(sl_1, sl_2) \equiv \text{Slope}(m_1, sl_1) \neq \text{Slope}(m_2, sl_2) \land m_1 \neq m_2 \land \text{IntersectingPoint}(p, sl_1, sl_2) \]

\[ \text{Between}(p.x, sl_1.p_1, x_1, sl_1.p_2, x_2) \land \text{Between}(p.x, sl_2.p_1, x_1, sl_2.p_2, x_2) \]

\[ \text{ProperCross}(sl_1, sl_2) \equiv \text{Cross}(sl_1, sl_2) \land \text{IntersectingPoint}(p, sl_1, sl_2) \land \text{ProperBetween}(p.x, sl_1.p_1, x_1, sl_1.p_2, x_2) \land \text{ProperBetween}(p.x, sl_2.p_1, x_1, sl_2.p_2, x_2) \]

In this operation, we have one special case when the intersecting point is also one of the endpoints of the line. In other words, one of the endpoints of a line lies on the other line. We will call this case of operation TCross (see Figure 6(f)).

\[ \text{TCross}(sl_1, sl_2) \equiv (\exists p)((\text{Endpoint}(p, sl_1) \land \text{Ontheline}(p, sl_1)) \lor \text{Endpoint}(p, sl_2) \land \text{Ontheline}(p, sl_2)) \]

4. Overlap

Two single lines are overlapped if and only if their slopes are equal and there is one endpoint lies in another line (see Figure 6(g)).

\[ \text{Overlap}(sl_1, sl_2) \equiv \text{Slope}(m_1, sl_1) = \text{Slope}(m_2, sl_2) \land m_1 = m_2 \land (\exists p)((\text{Endpoint}(p, sl_1) \land \text{Ontheline}(p, sl_1)) \lor (\text{Endpoint}(p, sl_2) \land \text{Ontheline}(p, sl_2))) \]

5. Within

Considering Within operation, we can divide it into two more types: CompleteWithin and SharedEndpointWithin (see Figure 6(h,i)).

\[ \text{CompleteWithin}(sl_1, sl_2) \equiv \text{Slope}(m_1, sl_1) \land \text{Slope}(m_2, sl_2) \land m_1 = m_2 \land (\forall p)((\text{Endpoint}(p, sl_1) \land \text{Between}(p.x, sl_2.p_1, x_1, sl_2.p_2, x_2)) \lor (\text{Endpoint}(p, sl_2) \land \text{Between}(p.x, sl_1.p_1, x_1, sl_1.p_2, x_2))) \]

\[ \text{SharedEndpointWithin}(sl_1, sl_2) \equiv \text{Slope}(m_1, sl_1) \land \text{Slope}(m_2, sl_2) \land m_1 = m_2 \land (\exists p_1, p_2)((\text{Endpoint}(p_1, sl_1) \land \text{Endpoint}(p_2, sl_1)) \lor (\text{Endpoint}(p_2, sl_2) \land \text{Endpoint}(p_1, sl_2))) \lor \text{Between}(p_2.x, sl_1.p_1, x_1, sl_1.p_2, x_2)) \lor \text{Between}(p_2.x, sl_1.p_1, x_1, sl_1.p_2, x_2)) \]

Figure 6. Spatial Operations
The relationship among all three operations are as follows:

\[
\text{CompleteWithin}(sl_1, sl_2) \land \text{SharedEndpointWithin}(sl_1, sl_2) \land \text{Overlap}(sl_1, sl_2)
\]

2.2.4 Point and Connected line

In the following, we will discuss the formalization of spatial operations between point and connected line. Considering point and connected line, there are three possible spatial operations: Endpoint, InteriorEndpoint and Ontheline.

1. Endpoint

**Endpoint** (see Figure 6(i)) is one of the relationship between point and connected line. In the formalization, we also need to distinguish unambiguously the two endpoints (same as operations between Point and Single line) so we define two operations on a line: EndpointNe and EndpointSw. The directions will be considered in the same fashion.

A point \( p \) will be an endpoint of connected line \( cl \) if and only if \( p \) is equal to \( p_1 \) or \( p_N \) of the connected line \( cl \).

\[
\text{Endpoint}(p, cl) \equiv p = cl.p_1 \lor p = cl.p_N
\]

**EndpointNe(p, cl) \equiv EndpointNe(p, cl) \land \exists p \neq p_1 \land \land (cl.p \cdot y > cl.p_1 \cdot y) \lor (cl.p \cdot y = cl.p_1 \cdot y \land cl.p \cdot x < cl.p_1 \cdot x)
\]

**EndpointSw(p, cl) \equiv Endpoint(p, cl) \land \exists p \neq p_1 \land \land (cl.p \cdot y < cl.p_1 \cdot y) \lor (cl.p \cdot y = cl.p_1 \cdot y \land cl.p \cdot x > cl.p_1 \cdot x)
\]

The relationship between EndpointNe and EndpointSw can be specified as follows:

\[
\text{EndpointSw}(p, cl) \rightarrow \text{EndpointNe}(p, cl) \land \text{EndpointSw}(p, cl) \rightarrow [(cl.p_1 \cdot y > cl.p_2 \cdot y_2) \lor (cl.p_1 \cdot x = cl.p_2 \cdot x_2) \land cl.p_1 \cdot x > cl.p_2 \cdot x_2]
\]

2. InteriorEndpoint

**InteriorEndpoint** (see Figure 6(k)) is another relationship between point and connected line when a point falls in the joint between two single lines of connected line.

\[
\text{InteriorEndpoint}(p, cl) \equiv \exists p \in \{p_i \mid i \neq 1, N \} | p \neq p_1
\]

3. Ontheline

A point \( p \) is on the connected line \( cl \) if and only if \( p \) is on one of the single lines \( sl_i \) of connected line \( cl \) (see Figure 6(l)).

\[
\text{Ontheline}(p, cl) \equiv \exists i \in \{sl_i \mid sl_i = (p_i, p_{i+1})\} | \text{Ontheline}(p, sl_i)
\]

The relationship among all three operations are as follows:

\[
\text{Endpoint}(p, cl) \rightarrow \text{Ontheline}(p, cl) \land \text{InteriorEndpoint}(p, cl) \rightarrow \text{Ontheline}(p, cl)
\]

\[
\text{Endpoint}(p, cl) \equiv \neg \text{InteriorEndpoint}(p, cl)
\]

2.2.5 Single line and Connected line

1. Meet

A single line \( sl \) meets connected line \( cl \) when they share one of the endpoints (see Figure 6(m)).

\[
\text{Meet}(sl, cl) \equiv (\exists p)[\text{Endpoint}(p, sl) \land \text{Endpoint}(p, cl)]
\]

2. Cross

A single line \( sl \) and connected line \( cl \) are crossed if and only if there is a single line \( sl \) of connected line \( cl \) crossing the single line \( sl \) (see Figure 6(o)).

\[
\text{Cross}(sl, cl) \equiv (\exists p)[((\text{Endpoint}(p, sl) \land \text{Ontheline}(p, cl)) \land \neg \text{Endpoint}(p, cl)) \lor (\text{Endpoint}(p, cl) \land \text{Ontheline}(p, sl) \land \neg \text{Endpoint}(p, sl))] \lor (\text{InteriorEndpoint}(p, cl))]
\]

3. Overlap

A single line \( sl \) and connected line \( cl \) are overlapped when one of the single line \( sl_i \) of connected line \( cl \) and single line \( sl \) are overlapped (see Figure 6(n)).

\[
\text{Overlap}(sl, cl) \equiv (\exists sl_i \in \{sl_i \mid \text{Overlap}(sl_i, sl) \lor \text{Equal}(sl_i, sl_i))]
\]

4. Within

4.1 CompleteWithin

A single line \( sl \) is completely within a connected line \( cl \) if and only if a single line \( sl \) is completely within, shared-endpoint within or equal to a single line \( sl_i \) of connected line \( cl \) (see Figure 6(q)).

\[
\text{CompleteWithin}(sl, cl) \equiv (\exists sl_i \in \{sl_i \mid i = 1, 2, \ldots, N-1, N \} | [\text{CompleteWithin}(sl_i, sl_i) \lor \text{Overlap}(sl_i, sl_i) \lor \text{Equal}(sl_i, sl_i))]
\]

4.2 ShareEndpointWithin

A single line \( sl \) is considered as shared-endpoint within connected line \( cl \) if and only if they share an endpoint and a single line \( sl \) is shared-endpoint within or equal to single line \( sl_i \) or \( sl_{i+1} \) of connected line (see Figure 6(t)).

\[
\text{ShareEndpointWithin}(sl, cl) \equiv (\exists sl_i \in \{sl_i \mid i = 1, N-1 \} | \text{ShareEndpointWithin}(sl_i, sl_i) \lor \text{Equal}(sl_i, sl_i)) \land (\exists p)[\text{Endpoint}(p, sl) \land \text{Endpoint}(p, cl)]
\]

2.2.6 Connected line and Connected line

1. Equal

Two connected lines are equal if and only if they have exactly the same list of ordered points.

\[
\text{Equal}(cl_1, cl_2) \equiv (\forall p_i \in \{cl_1.p_i \land cl_2.p_i \mid p_i = p_j; if i = j)
\]

2. Meet

A connected line \( cl_i \) meets connected line \( cl_j \) when they share one of the endpoints (see Figure 6(s)).

\[
\text{Meet}(cl_1, cl_2) \equiv (\exists p)[\text{Endpoint}(p, cl_1) \land \text{Endpoint}(p, cl_2)]
\]

3. Cross

A connected line \( cl_i \) and connected line \( cl_j \) are crossed if and only if there is a single line \( sl_i \) of connected line from both connected lines crossing each other (see Figure 6(u)).

\[
\text{Cross}(cl_1, cl_2) \equiv (\exists sl_i \in \{sl_i \mid sl_i \subset cl_1 \land sl_i \subset cl_2 \})
\]
A special case of cross is when one of the endpoints of one line lies on another line but not at the endpoint (see Figure 6(v)).

\[ T \text{Cross}(c_1, c_2) \equiv (\exists p)((\text{Endpoint}(p, c_1) \land \text{Ontheline}(p, s_{l_2})) \lor (\text{Endpoint}(p, c_2) \land \text{Ontheline}(p, s_{l_1}))) \]

4. Overlap

An connected line \( c_1 \) and connected line \( c_2 \) are overlapped when their single lines are overlapped (see Figure 6(i)).

\[ \text{Overlap}(c_1, c_2) \equiv (\exists s_{l_1} \in c_1, s_{l_2} \in c_2, s_{l_1} \neq s_{l_2}) [\text{Overlap}(s_{l_1}, s_{l_2}) \lor \text{Equal}(s_{l_1}, s_{l_2})] \]

5. Within

5.1 CompleteWithin

A connected line \( c_1 \) is completely within a connected line \( c_2 \) if and only if all single lines of connected line \( c_1 \) is completely within the connected line \( c_2 \) (see Figure 6(w)).

\[ \text{CompleteWithin}(c_1, c_2) \equiv (\forall s_{l_1} \in c_1, s_{l_1} \in s_{l_2}) [\text{CompleteWithin}(s_{l_1}, s_{l_2})] \]

5.2 ShareEndpointWithin

A connected line \( c_1 \) is considered as shared-endpoint within connected line \( c_2 \) when only the connected line \( c_1 \) is CompleteWithin the connected line \( c_2 \) but also share one of the endpoints (see Figure 6(x)).

\[ \text{ShareEndpointWithin}(c_1, c_2) \equiv \text{CompleteWithin}(c_1, c_2) \land (\exists p) [\text{Endpoint}(p, c_1) \land \text{Endpoint}(p, c_2)] \]

3. OWL/Protege Realization

Protege [Rubin et al. 2007] is a well-known and widely-used open-source platform for ontology management including creation, visualization, and manipulation of ontology [Horridge 2004]. Moreover, Protege is also user-friendly and domain-customizable because it allows user-defined or imported plugins.

3.1 Protege Background & Preparation

Our work uses Protege-OWL 3.4.4 which comes with SWRL Tab supporting reasoning and query. We also use Java JDK version 1.5.0.11 and Jess® rule engine [Sandia National Laboratories 2011] to do the inferences. In Protege-OWL 3.4.4, there are three types of properties: object type property, data type property, and annotation property. Object type property is used to define a relationship between individuals and individuals. Data type property is used to define relationship between individuals and data literals such as integer, string, etc. Annotation property is used to attach metadata to classes, individuals, or properties [Drummond N and Horridge M 2005]. Protege also allow us to customize a property such as functional, non-functional, symmetric, or transitive. If a property is functional, a cardinality between subject and object is N:1; for example, hasBirthday property is functional because one person can have only one birthday whereas the same birthday can belong to many people. In contrast, if a property is non-functional, the cardinality between subject and object is 1:N; for example, hasISBN property is non-functional because a book can have multiple ISBNs whereas an ISBN can belong to only one book. If a property is symmetric, both following statements are true: subject-property-object and object-property-subject; for example, isFriendOf property is symmetric because if John is a friend of David, then a David is also a friend of John. If a property is transitive, the following statement: subject-property-object2 is true if subject-property-object1 and object1-property-object2 are true; for example, subclassOf property is transitive because if class A is a subclass of class B and class B is a subclass of class C, then class A is also a subclass of class C.

3.2 Method to Add Spatial Dimension to Ontology

To add a spatial dimension to ontology, we use our formal specification of spatial ontology and adopt one of the approaches of how [O’Connor and Amar 2010] add temporal dimension to existing ontologies with modifications of the approach to better fit in spatial ontology case. The following diagram gives an overview of how our approach looks like.

![Figure 7](https://example.com/figure7.png)

**Figure 7.** A method to add spatial dimension to existing ontologies

According to Figure 7 above, SProposition2D class has hasGeoShape relationship with SGeometry class which contains geometry types: point, single line, connected line, non-ring connected line, and ring connected line. Point consists of two numbers: longitude and latitude. Single line consists of exact two points. Connected line consists of three points or more. There are two types of connected lines: non-ring and ring. The difference between non-ring and ring connected lines is that the starting point and ending point of non-ring connected line are not the same whereas the ones of ring connected lines are. hasDistance is a relationship between SGeometry class and DistanceUnit class. DistanceUnit contains units of distance measurements: millimeter, centimeter, inch, foot, yard, meter, kilometer, and mile which allow flexibility in presentation on spatial ontology.

The principle is that any class in need of spatial dimension will be added as a subclass of SProposition2D class. Consequently, the class will automatically have hasGeoShape property that enables a class entity to be modeled as one of the spatial data types.
An example is shown in the Figure 8. Suppose we have a yellow page ontology which contains contact information of individuals, businesses, parks, trains rails, roads and highways, etc. We would like to add a spatial dimension to it. As a result, we will add class houses, parks, train rails, roads and highways to be subclasses of SPropostion2D class as shown in Figure 7 so that those classes will have spatial features of one of the spatial data types. Intuitively, houses and schools are modeled as points, roads and highways are modeled as connected lines, and parks are modeled as polygons. The choice of geometry objects depend on the users and application scenarios.

To translate this method to Protégé, we have done the followings. A point is defined as latitude(x) and longitude(y) coordinate. The relationship of latitude(x) is defined by datatype property hasX carrying float value and the relationship of longitude(y) is defined by datatype property hasY carrying float value. hasX and hasY are functional properties because one point can have only one x and one y value respectively whereas a float number can belong to multiple points. Point is used to model an entity which does not have area. A single line is defined by exact two points and they cannot be the same. hasP1 and hasP2 are functional object type properties of a single line carrying first and second points respectively. A connected line is defined as a set of three or more ordered points. hasList is a functional data type property of a connected line. hasList carries string literal as an object, which has a following string pattern.

\[ \{n, p_1, p_2, ..., p_n\} \]

where n is number of points, n \geq 3 and p_i is a point i. This string pattern will be eventually parsed by Protégé built-ins into number of point’s value and point instances. A non-ring connected line is used to model a line with multiple edges such as long highways or train rails that do not have the same starting and ending points. A ring connected line is defined the same fashion as the non-ring connected line but its starting point and ending points are the same. To make it convenience in sharing spatial built-ins between non-ring and ring connected lines in Protégé, we define a ring connected line as non-ring connected line such that a set of ordered points of the ring connected line will have additional member which is the starting point as a last member of the set. As a result, a set of ring connected line will contain at least four points where the first and last members of the set are the same as opposed to three points in the reality. For example, a ring connected line of p_1, p_2, p_3 will be defined as \( \{4, p_1, p_2, p_3, p_1\} \). A ring connected line will have the same hasList property as a non-ring connected line, which carries string literal.

### 3.3 Developing Spatial Built-ins in Protégé

According to Figure 3, to allow us to do spatial reasoning, inference, and query in Protégé, we also develop spatial built-ins. We refer to our formalization of spatial operations to develop spatial built-ins in Protégé. We implemented 33 major spatial built-ins along with additional 6 utility built-ins in Protégé as shown in Figure 9.

Spatial built-ins can be divided into six categories depending on the combinations of geometry data types: point versus point, point versus single line, point versus connected line, single line versus single line, single line versus connected line, and connected line versus connected line. At this point, polygon data type will be left for future works.

<table>
<thead>
<tr>
<th>Spatial Built-ins</th>
<th>Point(1)</th>
<th>Single Line(2)</th>
<th>Connected Line(3) [Non-Ring &amp; Ring]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point(1)</td>
<td>Equal</td>
<td>Endpoint</td>
<td>Endpoint*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EndpointNe</td>
<td>EndpointSw*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ontheline</td>
<td>Interpoint</td>
</tr>
<tr>
<td>Single Line(2)</td>
<td>Equal</td>
<td>Meet</td>
<td>Meet*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cross</td>
<td>Cross</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TCross</td>
<td>Overlap</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Overlap</td>
<td>- Complete</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Within:</td>
<td>- SharedEndpoint</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Complete</td>
<td>- SharedEndpoint*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Intersect</td>
<td>Intersect</td>
</tr>
</tbody>
</table>

*Operations apply only on non-ring connected lines.

- a) 33 major spatial built-ins
- b) 6 additional spatial utility built-ins

### Figure 9. Show major 30 built-ins

All spatial built-ins mentioned in the Figure 9 are developed by this work and followed by the spatial formalization in Section 2. We make two assumptions to use the built-ins. First, 1 represents point, 2 represents single line, and 3 represents connected line: non-ring and ring. Second, when defining a spatial rule in Protégé, the built-ins have to be specified as following pattern:

spatial: <SpatialBuilt-insName>XY(<GeoDataTypeX>,<GeoDataTypeY>)

where X and Y are ordered number corresponding to geometry data types.

For example, to define an EndPoint built-in of point and single line, we will specify as:

```
Additional Built-ins | Slope, IntersectPoint, Between, ProperBetween, Distance (in km.), GetDataTypePropertyValue
```

7
The reason behind this is because different combinations of geometry data types have different ways of calculation on the same operation. As a result, we need to indicate what exact operation we are using.

3.4 Spatial Reasoning, Inference and Querying

Once we have a spatial dimension added to the ontology and have spatial built-ins ready in Protégé, we can now do spatial reasoning, inference and querying.

![Figure 10](image)

**Figure 10.** An example of how spatial operations can be used in reasoning and querying

We do the spatial reasoning by defining rules in SWRL Tab and use Jess® [Sandia National Laboratories 2011] rule engine to do the inference on those rules. Spatial built-ins that we developed can incorporate with other built-in libraries including SWRLB, TEMPORAL, SQWRL, etc. to create complicated rules.

The following example shows how spatial built-ins can be used in reasoning and inference. Suppose we have 2 houses, 1 school and 2 roads as illustrated in Figure 10. We would like to define a rule regarding the school zone. Suppose, we define a location in school zone as a point within 3 kilometers radius centered at a school. We can define as following:

$$\text{Point}(\text{p}) \land \text{hasX}(\text{p}, 3, \text{x}) \land \text{hasY}(\text{p}, 3, \text{y}) \land \text{School}(\text{s}) \land \text{hasX}(\text{s}, \text{x}) \land \text{hasY}(\text{s}, \text{y}) \land \text{spatial:distance}(\text{r}, 3, \text{x}, \text{y}) \land \text{swrlb:lessthan}(\text{r}, 3) \rightarrow \text{SchoolZonePoints}(\text{p})$$

For the next example, suppose we would like define highway-connected local roads as local roads which intersect with any highway at some point. Therefore, we will define highway-connected local roads as:

$$\text{Roads}(\text{r}) \land \text{hasList}(\text{r}, 3) \land \text{Highways}(\text{h}) \land \text{hasList}(\text{h}, 3) \land \text{spatial:Intersect33}(\text{r}, \text{h}) \land \text{spatial:cross33}(\text{r}, \text{h}) \land \text{spatial:intersect33}(\text{r}, \text{h}) \rightarrow \text{HighwayConnectedLocalRoads}(\text{r})$$

After we use Jess® processed the above rules, the qualifying point ?p and the road ?r will be categorized as an instance of SchoolZonePoints and HighwayConnectedLocalRoads classes.

To do the querying, we borrow a select built-in select from SQWRL library. The following example (see Figure 11) shows how spatial operations can be used in querying functionality. Suppose we have 3 roads in our ontology defined as $r1=[3, 1, 1, 3, 9, 6]$, $r2=[4, 2, 5, 8, 2, 6, 7, 9, 5]$ and $r3=[3, 1, 4, 1, 6, 3, 9]$ and would like to list all road intersections (x,y). So, the query can be defined as:

$$\text{Roads}(\text{r}) \land \text{hasList}(\text{r}, 3) \land \text{Roads}(\text{r2}) \land \text{hasList}(\text{r2}, 3) \land \text{spatial:cross33}(\text{r}, \text{r2}) \land \text{spatial:intersect33}(\text{r}, \text{r2}) \land \text{spatial:intersect33}(\text{r}, \text{r1}) \land \text{spatial:intersect33}(\text{r}, \text{r2}) \rightarrow \text{sqwrl:select}(\text{x}, \text{y})$$

![Figure 11](image)

**Figure 11.** How to define rule in Protégé

Once we processed the above query, the following is a resulting table from Protégé’s SQWRL Query Tab as shown in Figure 12.

![Figure 12](image)

**Figure 12.** Resulting window of query in Protégé

4. RELATED WORKS

4.1 Temporal Ontology Formalization

[Hobb 2007] presented temporal ontology formalization which can be embedded to OWL to capture temporal reasoning on the semantic web. There are four concepts that are mainly discussed on the paper: temporal relation reasoning, measuring duration, clock and calendar, and describing time and duration.
Temporal relation concept defines the reasoning of time instant and time interval such that time instant stores a point in time whereas time interval stores starting and ending times. In addition, the temporal operations reasoning are also discussed such as before, inside, timetbetween, properbetween, equal, meet, overlap, etc. Measuring duration concept defines a model to measure the unit of time, converts one unit of time to another. Hath and concatenation are also mentioned. Clock and calendar concepts are used to appropriately reason time zone and handle months with different days such as 28 days in the month of February. Describing time and duration concepts are used to capture time and duration with different kinds of format such as timestamp.

In the area of spatio-temporal ontology, temporal ontology formalization was already introduced. However, in the case of spatial ontology, although spatial concepts and operations are well-known, there are few attempts at specifying a complete formal ontology for spatial concepts. Our work refers [Hobb 2007]'s paper as a foundation to built upon and extended to two dimension with 360 directions in term of spatial operations. We also add more data types other than just point and interval. Additionally, spatial ontology also involves distance and area with different units of measurements such as millimeter, centimeter, inch, and kilometer.

4.2 Temporal Information Representation and Querying in OWL

[O'Connor and Amar 2010] proposed a technique to represent time dimension in OWL and demonstrated how temporal reasoning and querying over those ontologies on Protégé can be done.

[O'Connor and Amar 2010] presented two approaches in modeling temporal dimension: via user-defined property and via super class relationship. With regard to the first approach, they directly specify spatial features of entity through user-defined property. With regard to the second approach, to represent time in OWL on Protégé, they first create a Proposition class which has a relationship hasValidTime to ValidTime class. ValidTime class contains two types of time instances: time instant and time interval. Then, when we want to add temporal features to any class, the class will be added as a subclass of Proposition class and automatically, the class will eventually have time characteristics in it. [O'Connor and Amar 2010] also mentioned that the second approach is more powerful for two main reasons. The first reason is the ease in distinguishing the class with and without temporal features just by looking at which classes are subclasses of Proposition class. The second reason is the eligibility to have additional co-existing spatial representation in the future through multiple inheritance, by having more than one parent classes, without causing any effects and deleting the prior ones. As a result, in this work the second approach was selected over the first approach in adding temporal components into an existing ontology. The technique they used is known as “lightweight”. That is, time dimension could be easily added to existing ontologies with minimal changes. Once time dimension added, we can do temporal reasoning on Protégé via SWRL Tab. The built-ins temporal operations were implemented based on [Allen paper]. To do the querying, they borrow one of the built-ins of SQWRL library: select and use SQWRL Query Tab.

To represent spatial dimension in Protégé, there are some parts from [O’Connor and Amar 2010] we can adapt it to our works. However, some components need to be added and modified to better fit to our case; for example, hasTime, hasStartTime and hasFinishTime. Their all three relationships are not object type properties while in our case, single line and connected line the properties of which (hasP1, hasP2, and hasList respectively) are all object type property. As a result, we come up with a different approach in modeling spatial dimension to ontology. In term of spatial built-ins are also very much different, the temporal operations mostly deal with two directions at maximum whereas spatial operations deal with 360 degree directions most of the time. However, in term of query, we also use built-in “SQWRL:select” to do querying the same as the theirs.

4.3 Spatial Ontology Development

[Spaccapietra et al 2004] indicates the current status of spatial ontology research that needs attention. They also proposed a potential technique (MADS concept) that can be extended to deal with spatial ontology. MADS is a technique to allow regular database capture temporal and spatial characteristics by using their own temporal and spatial data types. MADS is also approved by consortium to be the best solution to the problem. There are many signs that the concept and knowledge of MADS can be adapted to solving spatial ontology management problem.

Our paper has different approach in capture the spatial reasoning on ontology. Our work starts with formalizing spatial concepts and operations ontology. We then introduce the lightweight technique similar to [O’Connor and Amar 2010] to add spatial dimension to existing ontologies in need to spatial dimension. We also develop the spatial built-ins in Protégé so that they can be used to do spatial reasoning. Finally, we use SWRL Tab, Jess rule engine, and SQWRL Query Tab to process the rules to do the spatial reasoning, inference, and querying.

5. CONCLUSION AND FUTURE WORKS

5.1 Conclusion

In conclusion, this paper presents spatial ontology formalization and a method to add spatial dimension to existing ontologies. Then, develop spatial built-ins to be used in spatial reasoning on Protégé. To process reasoning and inference we use SWRL Tab and Jess® rule engine. We also use select built-in from SQWRL library along with SQWRL Query Tab to do querying. With all components presented in this paper, they allow us to be able to add spatial dimension to ontologies through Protégé framework and make them to be able to capture and reason spatial meaning over semantic web more efficiently, productively and realistically.

5.2 Future works

This paper is an initial process which does not include polygon data type. In the future, we will improve our model to be able to capture a spatial polygon data type and also develop spatial built-ins for it in Protégé. In addition, we will continue to define formalizations of distance and area measurements. Finally, we
also would like to develop a technique by which the spatial ontology formalization can play a role as a medium to convert one spatial ontology from one system to another.

6. REFERENCES


[28] Spaccapietra, Stefano et al. "On Spatial Ontologies." Database Laboratory, Swiss Federal Institute of Technology, Lausanne, Switzerland: 2004