UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

VÍTOR FORTES REY

A positioning ontology for C-SLAM

Monograph presented in partial fulfillment
of the requirements for the degree of
Bachelor of Computer Science

Prof. Dr. Edson Prestes
Advisor

Porto Alegre, December 6th, 2013

CIP – CATALOGING-IN-PUBLICATION

Vítor Fortes Rey,

A positioning ontology for C-SLAM /

Vítor Fortes Rey. – Porto Alegre: Graduação em Ciência da Computação da UFRGS, 2013.

56 f.: il.

Monograph – Universidade Federal do Rio Grande do Sul. Curso de Bacharelado em Ciência da Computação, Porto Alegre, BR–RS, 2013. Advisor: Edson Prestes.

1. Ontology. 2. Robotics. 3. Map merging. 4. SLAM. I. Prestes, Edson. II. Título.

Draft 1.0 December 16, 2013
vfrey@inf.ufrgs.br
http://inf.ufrgs.br/∼vfrey

ACKNOWLEDGMENTS

I would like to thank my family, that always supported me. My sister, for helping me with the drawings and, of course, my orientator Dr. Edson Prestes, for introducing me to the hard problems of robotics. A special thanks is in order for Mara Abel, for years of ontology training.

CONTENTS

LIST OF FIGURES

ABSTRACT

Using ontologies it is possible to exchange information formally and precisely. In the domain of simultaneous localization and mapping, where a robot has to determine his location while he maps an unknown environment, different devices, that use different techniques to map the ambient, can communicate using ontologies and thus cooperate. This work presents an ontology based approach that enables this cooperation to happen. This is done extending existing ontological spatial representations and introducing a very adaptable software framework for representing spatial primitives and performing spatial reasoning.

Keywords: Ontology, Robotics, Map merging, SLAM.

ABSTRACT

Utilizando ontologias é possível compartilhar informações de maneira formal e precisa. No domínio da localização e mapeamento simultâneos, onde um robô tem de determinar sua localização enquanto mapeia um ambiente desconhecido, diferentes dispositivos, que usam diferentes técnicas para mapear o ambiente, podem se comunicar utilizando ontologias e assim cooperar. Este trabalho apresenta uma abordagem ontológica que possibilita essa cooperação. Para tal fim estendemos representações espaciais ontológicas existentes e introduzimos uma arquitetura de software altamente adaptável para a representação de primitivas espaciais e raciocício espacial.

Keywords:

.

Ontologias, Robótica, superposição de mapas, SLAM

1 INTRODUCTION

1.1 Motivation

As we have seen in (ARNDT et al., 2013), robotic systems can be part of complex, smart environments. As the systems become complex and ubiquitous, they start to interact more with both machine and men. In this connected future it is necessary that all parts communicate using a well-defined vocabulary with precise meaning so that they can achieve mutual understanding.

The process of formalizing the domain of robotic and automation was started in (SCHLENOFF et al., 2012) with the CORA ontology. Since the notion of space and, specially, of movement in space, is essential in the domain of robotics, CORA has a positioning extension presented in (J. L. CARBONERA S. R. FIORINI, 2013). This work is built upon their work, extending the concepts of positioning so that they can be used for generic map merging and thus SLAM.

1.2 Structure of this work

This work is divided into six more parts. In chapter 2, we give the conceptual basis for the work. In chapter 3, we create the extensions needed to represent map merging by simply extending the positiong in CORA. We will show the implementation of this extension in chapter 4. Some results for this implementation follow in chapter 5. In chapter 6, we propose diachronalize positioning in CORA in order to make it suitable for C-SLAM. At last, in chapter 7, we have the conclusion and future work.

2 CONCEPTUAL BASIS

In this chapter we will give the conceptual basis for this work, explaining SLAM, ontologies and spatial representations in general.

2.1 SLAM and the map merging problem

SLAM means Simultaneous Localization and Mapping. It is the problem of building a map while dealing with uncertainty in localization(ZHOU; ROUMELIOTIS, 2006). This is often the case for robots exploring unknown regions like bottom of the ocean, aiding in rescue sites or performing land mine extractions.

The map generated by SLAM can be used, during and after the process, for navigation, motion and mission planning(CARPIN; BIRK; JUCIKAS, 2005). The idea is that the robot, equipped with sensors, applies some algorithm to the sensed data to build an accurate map. It is assumed that the environment to be mapped possesses features such as doors, walls and other objects that can be easily identified by the robot.

Another problem of equal importance and enhanced complexity is C-SLAM, meaning Cooperative Slam, an instance of SLAM where multiple robots cooperate to build a merged, global map. But map merging is also useful in traditional SLAM, as a single robot, due to the errors in odometry, might need to merge its recently mapped area to the old one when finishing a loop.

Map merging is a hard problem with many algorithms proposed such as (CARPIN; BIRK; JUCIKAS, 2005) and (ZHOU; ROUMELIOTIS, 2006).

Our objective, in this work, is to represent formally spatial information and map merging in the domain of C-SLAM so that different robots can share spatial information and thus cooperate. To this purpose, we will not focus on merging maps to correct sensor errors, but on the merging of correct positioning data obtained by different robots. To represent formally spatial information and map merging, we will use ontologies.

2.2 Ontologies

The term “ontology” itself has different meanings in different fields. In this work we will use the definition that states that an ontology is “an explicit specification of a shared conceptualization”(STUDER; BENJAMINS; FENSEL, 1998).

In more details:

- The specification is explicit, meaning that all the information is formally defined in the ontology. In this definition lies one of the basic strengths of ontologies, as being explicit means they can be understood (and therefore used) by both machines and men.

- The ontology is a specification of a conceptualization, meaning it models the concepts we have of things rather than the things themselves. This means that an ontology about persons might have a concept for "Man", but it never possesses a concept for "Socrates".

- The conceptualization is also shared, meaning it complies with the intended meaning a community shares about the concepts and not that of a single individual. This means the ontology constitutes a shared meaning on a vocabulary.

2.2.1 Representing ontologies

An ontology is usually made of a taxonomy of concepts, a set of relations and axioms (PRESTES et al., 2013) represented in some logic-based language. As an example, let's consider a simple ontology about persons. In this ontology the concept of person might be represented by the *first-order logic* unary predicate $Person(x)$. So, to say that there exists person, we write

$$\exists x Person(x).$$

We might want to divide persons between men and women and so create concepts for them in the same manner. Since every man and woman is a person, it implies that

$$\forall x Man(x) \rightarrow Person(x)$$

and

$$\forall x Woman(x) \rightarrow Person(x)$$

. This means that the concepts Man and Woman are subsumed by the concept Person in our taxonomy. We can think of properties as relations between concepts, such as

$$isFatherOf(x, y)$$

to represent that x is the father of y. Axioms represent statements always valid in the domain as "every person is either a man or a woman" that can be represented as

$$Person(x) \rightarrow (Man(x) \wedge \neg Woman(x)) \vee (\neg Man(x) \wedge Woman(x))$$

While this simple example represent the ontology in first-order logic, ontologies in applications are usually represented in less expressive languages. This is done in order to make reasoning decidable or simply reduce reasoning complexity.

2.2.2 OWL

The OWL (Web Ontology Language(MICHAEL K. SMITH; DEBORAH L. MCGUINNESS, 2004)) is a W3C recommendation for semantic web applications. It is based on description logic and comes in three flavors with varying degrees of expressibility. OWL-Lite provides a taxonomy and simple restrictions, prioritizing reasoning efficiency. OWL-DL extends OWL-Lite, keeping its reasoning decidability, but providing more expressiveness in detriment of reasoning efficiency. At last, OWL-Full prioritizes expressiveness, but lacks reasoning decidability.

Up to now, OWL's basic components are "classes", "object properties" and "datatype properties". Classes define the concepts modeled in the ontology, such as "Human". Object properties model the binary relations between concepts, such as "hasChild". Datatype properties, on the other hand, define relations between concepts and datatypes, e.g, saying that a human has an integer for his age. It is easy to see how OWL lacks expressibility. For example, n-ary relations are not allowed in the language and knowledge about knowledge, also called meta knowledge, is also not possible in most versions of the language. Meta knowledge is not possible because it is usually represented using reification, that is, representing knowledge as an individual and referring to it. An example of meta knowledge using reification would be to say that "a human believes that he has a child", that can be represented in first-order logic as $\exists x, yHuman(x) \wedge Believes(x, hasChild(x, y))$.

Owl's simple components, associated modeling tools like Protégé(KNUBLAUCH et al., 2004), APIs and ease of reasoning make it very popular. Since OWL documents are represented in RDF (another W3C standard) or XML, support for it is available in most triple stores. Many reasoners like Pellet(SIRIN et al., 2007) and the Jena reasoner are also available for it, making it easy to use and extend reasoning in OWL.

2.2.3 Standard Upper Ontology Knowledge Interchangeable Format

The Standard Upper Knowledge Interchangeable Format, or SUO-KIF for short, is a free and open source logic language that can represent any arbitrary sentence in the first-order predicate calculus (GROUP, 2013a). One of its advantages is that it has a declarative semantics, which means one can understand its formulas without the aid of any interpreter. This means that no tool other than a text editor is needed to correctly visualize or even create them. Moreover, since the language does not include XML tags, its models in textual format are smaller and easier to read than the ones described in a language that does include tags, like OWL.

In fact, understanding its formulas is easy for those familiar with first order logic. For example, the notion of a subclass, present in $\forall xHuman(x) \rightarrow Man(x)$ is expressed in SUO-KIF as (subclass Man Human). To represent the fact that john is a human, we say (instance john Human). The language also allows for the use of variables and the quantifiers "exists" and "forall". Variables are always started with a question mark, so to say that "there is a human that has a child", we write

```
(exists (?X) (?Y)
      (and
            (instance ?X Human)
            (hasChild ?X ?Y)
      )
```

)

in SUO-KIF. This is equivalent to the formula $\exists x, yHuman(x) \wedge hasChild(x, y)$ in first-order logic.

More information on SUO-KIF is available at (GROUP, 2013b). The full SUO-KIF language definition is available at (GROUP, 2013a).

2.2.4 Top, Core and Domain ontologies

In order to enhance compatibility between two related ontologies that were developed separately or simply to avoid rework, ontologies can be built in layers of descending level of generality. According to (PRESTES et al., 2013), an ontology can be classified regarding its generality level as:

Top-level ontologies define very general concepts like time, space, events and matter. Examples of top-level ontologies are SUMO(NILES; PEASE, 2001) and Dolce (GANGEMI et al., 2002).

Core ontologies define the concepts of some generic domain according to a top-level ontology. For example, it defines that, in the robotics domain, a robot is a physical object. So a core ontology defines terms that are not so general as space and time, but are common enough in their area that they see a lot of reuse. For example, the concept of robot in the many domains related to robotics and automated systems and the notion of gene in biology. Examples of core ontologies are CORA(PRESTES et al., 2013) and the Gene Ontology (ASHBURNER et al., 2000).

Domain ontologies define the concepts of the domain at hand according to a core ontology. For example, model the domain of car-washing robots or Cyanobacteria.

This hierarchy, while useful, is not mandatory. Semantic web applications, for example, often skip the first two levels and simply start defining the domain concepts. Domain ontologies can also not use a core ontology and use the top-level directly instead.

2.2.5 The SUMO ontology

The Suggested Upper Merged Ontology(NILES; PEASE, 2001), or SUMO for short, is an upper level ontology. It is open source and was created by merging publicly available ontologies. One of its main features is the inclusion of many other high level theories as James Allen's time axioms(ALLEN; HAYES, 1985), the top ontology of John Sowa(SOWA, 1995) and others. SUMO also has a mapping to Wordnet, a lexical database for the English language, making it very useful for works in linguistics.

2.2.5.1 Ontological decisions

SUMO separates entities in two major categories: entities with a position in space/time (Physical) and everything else (Abstract). For example, an "Object" is a physical entity while a "Number" is an abstract one. So, when a math teacher writes a number on a chalk board, the writing itself is a Physical thing, but the number it represents is an Abstract one.

Physical things are separated in objects and processes, with objects being the ordinary objects like a chair and processes being things that have temporal parts or stages like a party. This means SUMO follows an endurantist perspective instead of a perdurantist one. Since this choice guides all modelling using SUMO, an explanation is in order:

For an endurantist, an object keeps its identity through time and so, while some processes might change things about it, every part that is essential to it is always present. On the other hand, for a perdurantist, an object is composed of every temporal part it has at all times and so all things about it are indexed in time. An easy analogy is to think that perdurantists see things as regions in a 4D space while endurantists see them as 3D things that can change in processes.

2.2.5.2 Implementation

SUMO's primary language is SUO-KIF, but a version in OWL is available at (GROUP, 2013c). Since the OWL language is very limited compared to SUO-KIF, many of its axioms are left only as comments in the OWL version. For example, the axiom that says that "if an object fills a hole, it fills all parts of the hole" is described in SUO-KIF as

```
(=>
        (and
                (fills ?OBJ ?HOLE1)
                (properPart ?HOLE2 ?HOLE1)
        )
        (completelyFills ?OBJ ?HOLE2)
)
```

and cannot be correctly translated to OWL.

2.2.6 The CORA ontology

CORA is a core ontology for the domain of robotics and automation. It is being developed by the IEEE-RAS working group entitled Ontologies for Robotics and Automation (ORA). Its goal is "to develop a standard ontology and associated methodology for knowledge representation and reasoning in robotics and automation, together with the representation of concepts in an initial set of application domains"(SCHLENOFF et al., 2012).

2.2.6.1 Ontological decisions

As of today, CORA uses SUMO as its top-level ontology and defines a set of core concepts for the domain of robots and automation. One of its core decisions is to define a robot as a device that is also an agent. This means a robot must be an agent in space/time that serves as an instrument to a given process. It also means a robot must be a physical object as opposed to, for example, a software bot.

A hierarchy of CORAs concepts and how they fit with the ones in SUMO can be seen in figure 2.1. Since CORA is still in development, it is important to notice that we are using its version as presented in (PRESTES et al., 2013).

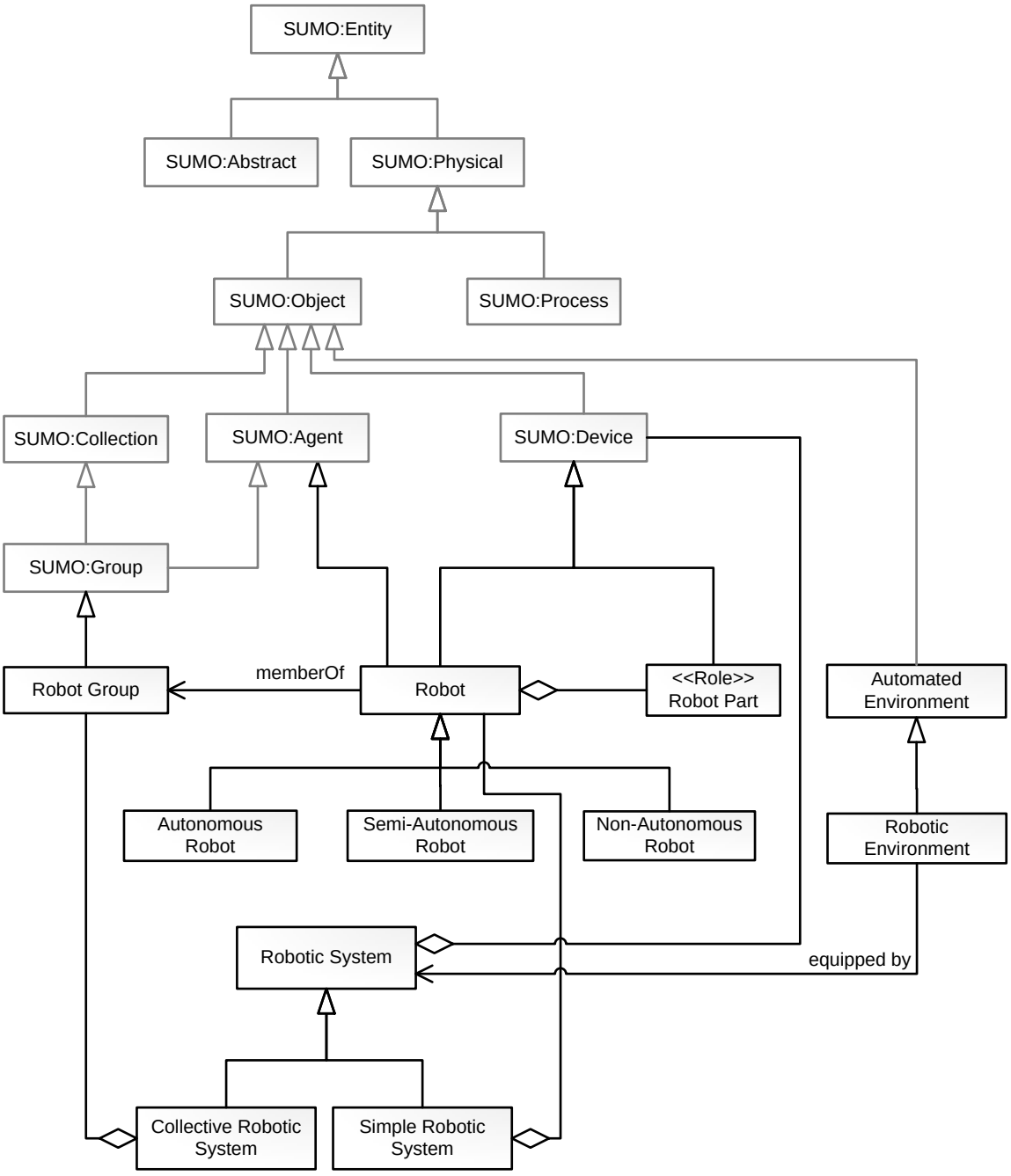

Figure 2.1: Concepts for CORA.

2.2.6.2 Implementation

The ontology was first defined in (SCHLENOFF et al., 2012). Later, a SUO-KIF version was made available at (CRAIG SCHLENOFF EDSON PRESTES, 2013).

2.3 Spatial Representations

To represent map merging and SLAM, first we need to understand what is spatial information and the different ways it can be represented. As said in (BATEMAN; FARRAR, 2006), the most basic ontological question about space is wether or not it exists independently of the things inside it. If it does not, space is a matter of inter-relationships between the existing objects. This is denominated the *relational* view of space, while the other is the *absolutist* view. SUMO, for example, adopts the *relational* view, as objects are located at regions, that are also objects in SUMO. In the following subsections some spatial attributes will be presented with their representation in SUMO explained.

2.3.1 Quantitative position representations

The quantitative position of something is its exact, quantified positioning that can be represented as a point in some coordinate system. The number of values the quantitative position of some object can assume is infinite. For example, the position of some object in $\mathbb{R}^1$ might be any number in $\mathbb{R}$.

While SUMO allows for the quantitative definition of many things, like time and temperature, it doesn't originally come with any general, quantitative spatial information for positioning. The closest thing it has might be the TernaryPredicate distance, that specifies the numeric, minimum distance between two objects. This notion, while useful, does not describe the *exact* quantified positioning of some object and so is not enough to represent the position of some object in $\mathbb{R}^2$, for example.

2.3.2 Qualitative position representations

Qualitative positioning assumes the relational view of space, representing objects always in relation to one another. A qualitative position is one that "can only take a small, predetermined number of values"(KLEER; BROWN, 1985). For example, an object can be described qualitatively as either near or far some other object. Many other qualitative ways of defining positioning can be defined. An object can be left or right another, inside or outside another, or even in some defined cardinal direction of another as seen in figure 2.2.

SUMO represents qualitatively the localization and orientation of objects in space using SpatialRelations. The *orientation* SpatialRelation defines the orientation of an object in relation to another as a PositionalAttribute, e.g. (orientation Ball1 Ball2 Above). The *localization* of the object is represented by the partlyLocated SpatialRelation that expresses that an object has at least one of its parts located at another. The located relation is a subrelation of partlyLocated and expresses that every part of the object resides at another. It is important to remember that a region is also an object in SUMO, so one could say, for example, (located Socrates Athens). The relation exactlyLocated is a subrelation of located and describes the *exact* location of the object, meaning that object $o1$

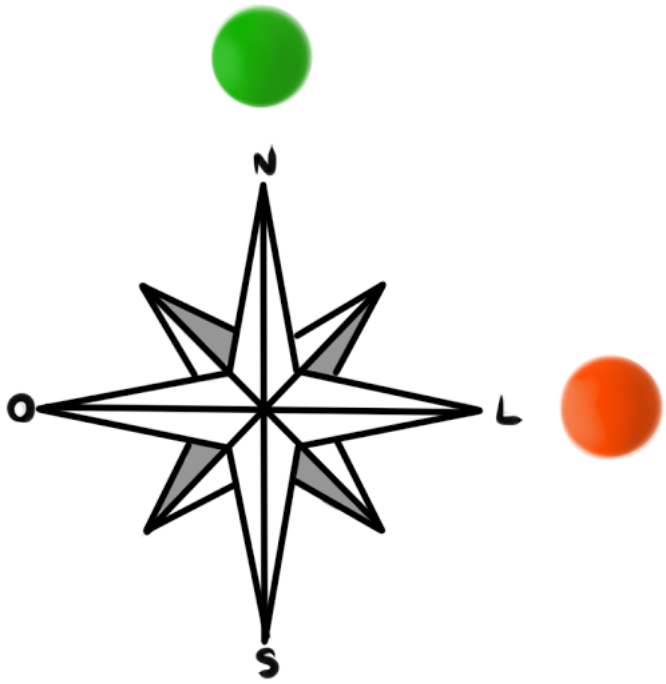

Figure 2.2: Cardinal directions can define qualitative positioning.

is exactlyLocated at object $o2$ and no other physical thing is exactlyLocated at $o2$ at that time.

2.3.3 Positioning extension for CORA

While SUMO contains qualitative positioning, it does not allow for the representation of precise, quantitative positioning information that is needed in the domain of robotics. To fill this gap, a positioning extension to CORA was proposed in (J. L. CARBONERA S. R. FIORINI, 2013). As it is part of CORA and so aims at representing spatial information for all the domain of robotics and automation, this ontology was chosen as the basis for all our spatial definitions.

The ontology is synchronic, meaning it represents only the spatial information at some instant and not how it changes as time passes. It has support for both qualitative and quantitative spatial information. Quantitative information is represented using points, while qualitative is represented using regions. In this extension, position and orientation are separated. Positioning is represented in PositionPoints and PositionRegions, while orientation is represented in OrientationPoints and OrientationRegions.

2.3.3.1 Coordinate Systems

One of the basic concepts of the ontology is the CoordinateSystem. Its function is to define a mapping of the point's coordinates to a numeric position in a subspace of $\mathbb{R}^n$. As a simple example, let's consider a square, two-dimensional world where the only thing that exists is a square box with side length one and the only valid positions are its edges.

If we represent the possible edges of the box using the set of tuples

$$sP = \{(DOWN, LEFT), (DOWN, RIGHT), (UP, LEFT), (UP, RIGHT)\},$$

a CoordinateSystem is a function that has as its domain sP and as its image the subset of $\mathbb{R}^2$

$$sR = \{(0,0), (0,1), (1,0), (1,1)\}$$

as we can see in figure 2.3.

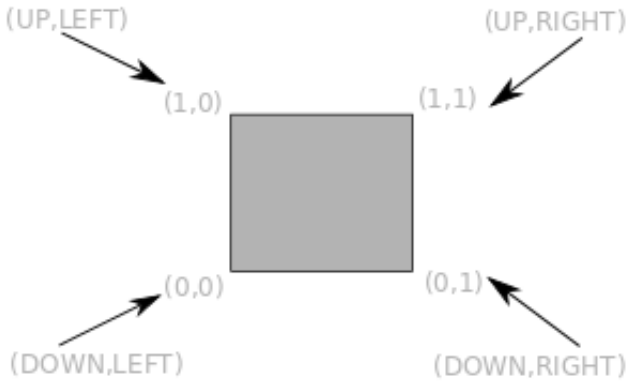

Figure 2.3: A simple CoordinateSystem example.

Following the relationist view of space from SUMO, a coordinate system in the positioning extension for CORA has always a reference SUMO object. That is, a coordinate system cannot exist in and on itself, but only in relation to something in space/time. The object the coordinate system exists in relation to can be its origin, but that is not obligatory. In this example the reference object of the coordinate system would be the box.

2.3.3.2 Positioning

As defined in (J. L. CARBONERA S. R. FIORINI, 2013), PositionPoints and PositionRegions are PositionMeasures, meaning they are measures (observations) attributed to a physical object. A PositionPoint is an exact position in a coordinate system while a PositionRegion is a region in it.

Every PositionPoint is in one and only one CoordinateSystem(CRAIG SCHLENOFF EDSON PRESTES, 2013). As it is an exact position, two objects cannot share the same PositionPoint at the same time. In this manner, using PositionPoints to describe the position of an object is similar to defining where it is exactlyLocated in SUMO. The difference is that a PositionPoint represents the quantitative location while exactlyLocated can only represent the qualitative one.

For example, let's consider positioning in a space that consists of an infinite, unidimensional strip of paper. One could represent PositionPoints in this space using a coordinate system that represents a position as a single number in $\mathbb{Z}$, as is shown visually at figure 2.4.

PositionRegions, on the other hand, are defined using objects. The purpose of regions in general will be explained later in section 2.3.3.5 when discussing qualitative positioning in the ontology.

Figure 2.4: Example of position in an unidimensional space. An object is positioned at -1.

2.3.3.3 Orientation

Analogous to positioning primitives, OrientationPoints and OrientationRegions represent points and regions in their own orientation coordinate systems. For example, an orientation point can be an unidimensional point representing in radians the angle between the direction of two robots.

Another simple example about orientation is the information a traveler can get from his compass. If a traveler has a compass like the one in figure 2.5, he does not know by it his exact position, only his orientation regarding the lines of the earth's magnetic field.

Figure 2.5: A compass pointing north can provide orientation.

OrientationRegions are analogous to PositionRegions and so will also be explained in section 2.3.3.5.

2.3.3.4 Pose

A pose, so common in the domain of robotics, is represented as the measure in position and orientation of some object. For example, let's consider the example of a compass in a grid as in figure 2.6. The coordinates of the grid represent the object's position while the angle of the compass can represent its orientation. Together, they form its *pose*.

2.3.3.5 Qualitative positioning

Qualitative positioning information is derived using operators. An operator takes an object as its parameter and returns a region over it. Operators can be about positioning or orientation, as PositionRegions and OrientationRegions are *always* generated by an operator. Thinking about positioning, to say, for example, that an object $o1$ is near another object $o2$, means that $o1$ is positioned in the region generated by applying the spatial operator "nearOf" to $o2$.

Coming back to the example given in 2.4, we can define the "nearOf" region of an

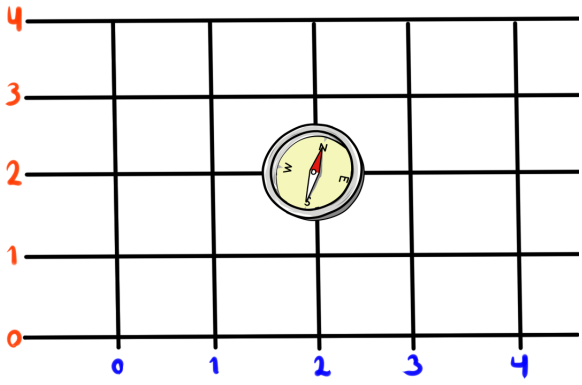

Figure 2.6: Position and orientation together forming a pose.

object in the unidimensional space as the PositionPoints that have a distance of at most 2 to it. This can be better seen in figure 2.7, where the yellow region is generated by applying the operator "nearOf" to the object in position -1. We can also consider using the operator "farOf", that when applied to the object will generate the blue region.

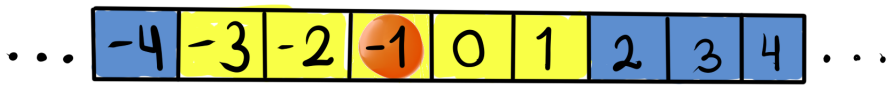

Figure 2.7: Generating a region by applying an operator to an object.

Analogously, one can say $o1$ is to the north of $o2$ using the orientation operator "toTheNorthOf". Going back to the example of the compass, we can define some simple regions and operators as seen in figure 2.8.

These operators are defined as SUMO functions, that are general mathematical functions, but often can be mapped to existing SUMO constructs like SpatialRelations.

2.3.3.6 Transformations

Transformations are mappings from the points of one coordinate system to another. While the ontology does not define that a transformation mapping $p1$ to $p2$ *means* that $p1$ is $p2$ in $p2$'s coordinate system, this is its common usage in the case of map merging.

Using transformations in this way one can translate the positions of one coordinate system to another, making data collected by different individuals, and even represented in different coordinate systems, interoperable. For example, using transformations a robot that uses some polar coordinate system could share knowledge with another that uses a Cartesian one. Even a flying drone that uses a $3D$ coordinate system could share the location of features with a land robot that uses a $2D$ one.

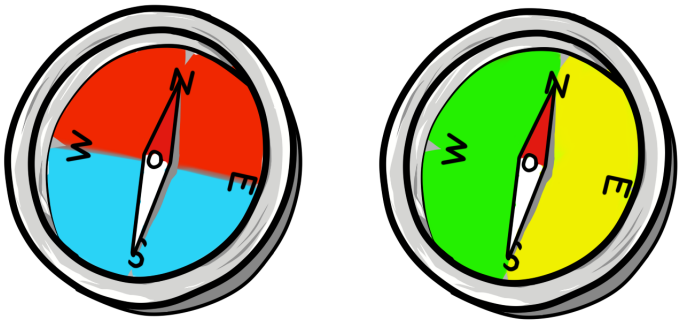

Figure 2.8: Example of OrientationRegions generated by operators. The operator "toTheNorthOf" generates the red one, "toTheSouthOf" the blue, "toTheWestOf" the green and"toTheEastOf" the yellow.

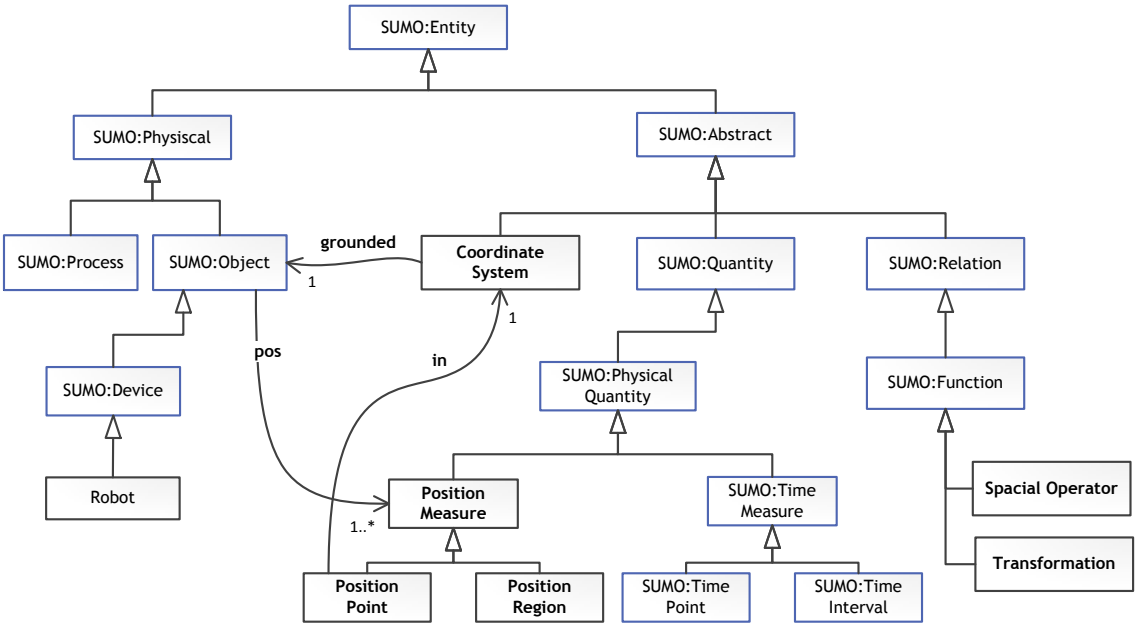

Figure 2.9: Main concepts for the positioning extension for CORA.

2.3.3.7 Implementation

The ontology was encoded first in first order logic at (J. L. CARBONERA S. R. FIORINI, 2013) and later in SUO-KIF at (CRAIG SCHLENOFF EDSON PRESTES, 2013). The main concepts of the ontology and their relations to concepts in SUMO can be seen in figure 2.9.

3 EXTENDING THE STATIC POSITIONING IN CORA

In this chapter we will introduce the extension to positioning in CORA that will be used to represent spatial information in the domain of C-SLAM. We will also formally define the process of map merging and present the method for obtaining the transformations it needs based on our positioning representations.

3.1 Representing positioning in the domain

In the position ontology defined in (J. L. CARBONERA S. R. FIORINI, 2013), as is expected of part of a core ontology, many commitments are left to the domain ontology. For example, in order to actually use the ontology in the domain, one must first define which are the types of positioning and orientation coordinate systems that will be used.

As is common in the domain of map merging, we will assume that the space to be mapped by the robots can be represented in two dimensions. For this purpose, we will define the coordinate system of a robot so that every point in it is represented as a 2-tuple

$$\eta(p) = (x(p), y(p))$$

where $(x(p), y(p))$ represent, respectively, the point's x and y coordinates in the cartesian canonical orthonormal base with standard orientation. In other words, we will use the vectors

$$V = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

with the first base pointing to the right as seen in figure 3.1 as the bases for our coordinate system. Each coordinate system will have a robot as its reference object so that the point $(0, 0)$ represents the robot's quantitative position in it.

We will name this coordinate system *CartesianCoordinateSystem*, or Ccs for short. Points in it will be *CartesianPositionPoints* and regions in it *CartesianPositionRegions*.

Regarding orientation, as is usually done in the domain, the y axis will point the orientation of the robot as seen in 3.2.

This means that, for any robot $r1$ that uses a CartesianCoordinateSystem to map the environment, its coordinate system base will be the canonical orthonormal base and any other robot $r2$ he meets that also uses a CartesianCoordinateSystem will have for its base

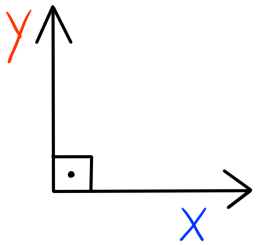

Figure 3.1: Base vectors for our coordinate system

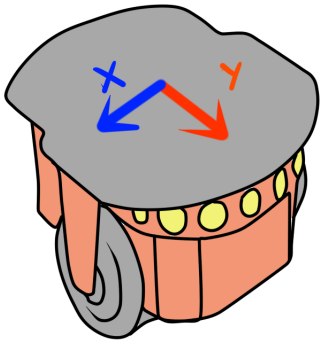

Figure 3.2: A robot with its coordinate system. The Y axes points to its orientation.

vectors $r1$s vectors rotated by some angle from 0 to 2π. The range of possible base vectors for $r2$ can be seen in figure 3.3.

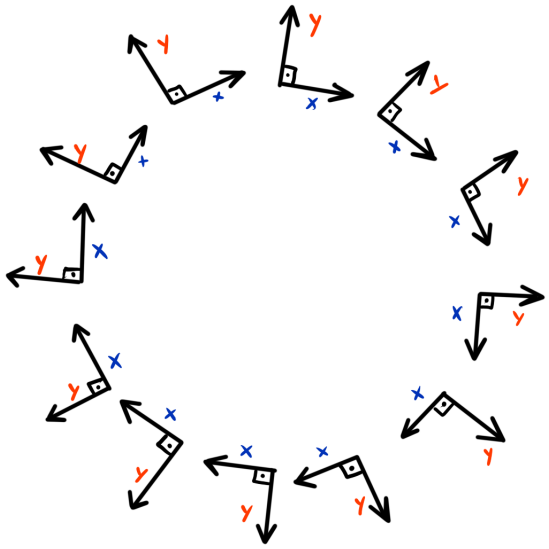

Figure 3.3: The possible base vectors of $r2$ according to $r1$.

Following [CORA POS]s decision to separate orientation from position, the orientation of the robot will have its own coordinate system. We will call it *AngularOrientationCoordinateSystem* or AngularCS for short. To attribute an *AngularOrientationPoint* with value ov to some object $o1$ in relation to $o2$ means that the Ccs base of $o2$ can become the base of $o1$ when multiplied by a rotation matrix of ov degrees. One example can be found in figure 3.4.

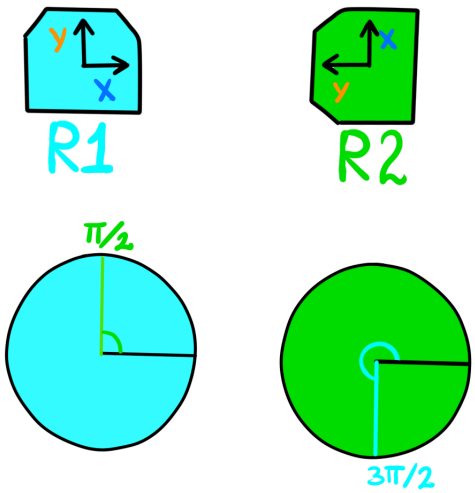

Figure 3.4: Orientation of R2 as seen by R1 and of R1 as seen by R2.

3.1.1 Formalization in SUO-KIF

Now that the mathematical notions used were explained, let's represent our primitives in SUO-KIF.

3.1.1.1 CartesianPositionPoint

A CartesianPositionPoint is a PositionPoint in a CartesianCoordinateSystem. Therefore:

```
(subclass CartesianPositionPoint PositionPoint)

(=>
        (instance ?P CartesianPositionPoint)
        (exists (?C)
                (and
                        (instance ?C CartesianCoordinateSystem)
                        (inCS ?P ?C)
                )
        )
)
```

We must also relate a point p to its unique coordinates $(x(p), y(p))$. This is encoded as:

```
(subclass hasX UnaryFunction)
        (domain hasX CartesianPositionPoint)
        (range hasX RealNumber)

(subclass hasY UnaryFunction)
        (domain hasY CartesianPositionPoint)
        (range hasY RealNumber)

(=>
        (instance ?P CartesianPositionPoint)
        (and
                (exists (?X) (hasX ?P ?X))
                (exists (?Y) (hasY ?P ?Y))
        )
)
```

3.1.1.2 CartesianCoordinateSystem

A CartesianCoordinateSystem is a type of CoordinateSystem that uses only CartesianPositionPoints. In SUO-KIF:

```
(subclass CartesianCoordinateSystem CoordinateSystem)
```

```
(=>
        (instance ?C CartesianCoordinateSystem)
        (forall (?P)
                (=>
                        (inCS ?P ?C)
                        (instance ?P CartesianPositionPoint)
                )
        )
)
```

It also places its reference object at the origin. This means:

```
(<=>
        (and
                (instance ?C CartesianCoordinateSystem)
                (ref ?C ?OBJ)
        )
        (and
                (pos ?OBJ ?P)
                (inCS ?P ?C)
                (hasX ?P 0)
                (hasY ?P 0)
        )
)
```

3.1.1.3 CartesianPositionRegions

CartesianPositionRegions will be the regions in our cartesian space. Formally:

```
(subclass CartesianPositionRegion PositionRegion)
(=>
        (instance ?R CartesianPositionRegion)
        (forall (?P)
                (=>
                        (inPR ?P ?R)
                        (instance ?P CartesianPositionPoint)
                )
        )
)
```

3.1.1.4 CartesianOperators

CartesianOperators are simply the operators that will generate our CartesianPosition-Regions.

```
(subclass CartesianOperator Operator)
```

(range CartesianOperator CartesianPositionRegion)

3.1.1.5 AngularOrientationPoint and AngularOrientationCoordinateSystem

The AngularOrientationPoint represents the orientation from one object to the other. Since, as of today, orientation is still not well defined in CORA, we will not define formally in SUO-KIF AngularOrientationPoints and AngularOrientationCoordinateSystems.

3.2 Representing generic static map merging

When a roboticist speaks of a map acquired in SLAM, he refers to some spatial information gathered by a robot, usually in the form of a set of points or regions. So, in the positioning extension for CORA, a map is no more than the spatial information in the robot's coordinate system and so to do a map merge is to unify the information of two maps in another one by the use of transformations.

Using first order predicate CS to represent a CoordinateSystem, predicate T to represent a Transformation and $mapsCS(c1, c2, t)$ to represent the existence of a mapping from CoordinateSystem $c1$ to CoordinateSystem $c2$ generated by Transformation t, we have that a merge of $c1$ and $c2$ into $c3$ is, formally:

$$\exists c3, t1, t2 CS(c3) \wedge T(t1) \wedge T(t2) \wedge mapsCS(c1, c3, t1) \wedge mapsCS(c2, c3, t2)$$

$$\equiv$$

$$merge(c1, c2, c3)$$

So, if a robots $r1$ wants to access robot $r2$'s map, we will choose $c1$ and $c3$ as the coordinate system of $r1$ and $c2$ as $r2$'s coordinate system, doing $merge(c1, c2, c1)$. In this case, since $t1$ can trivially map every point of $c1$ to itself, only the transformation $t2$ will need to be found. On the other hand, in C-SLAM we will want to choose as the merging destination some arbitrary global coordinate system cA and do $merge(c1, c2, cA)$, which means we will need to find both $t1$ and $t2$.

This definition of merge does not specify how the transformations that define the merge are to be obtained, neither does it assume that the mapping was done perfectly. As seen in (ZHOU; ROUMELIOTIS, 2006), a map merge *aims* at being a perfect merge, but this is often not feasible in real world applications. In fact, one could construct all possible merged maps using this definition and rank them according to their confiability. This will not be the case of this work, where the transformations will be generated only when the coordinate systems points can be mapped perfectly.

This definition of map merge is also very permissive regarding how the spatial information can be represented, as it accounts for merging of completely different maps that could have been obtained using completely different SLAM techniques.

3.3 Obtaining the transformations

As seen in section 3.2, to find a map merge of two robot's maps one must define how the transformations will be obtained. We will focus on perfect map merging, as we consider that the robot's readings are perfect in the sense that they *don't* include measurement errors.

Following the ideas presented in (ZHOU; ROUMELIOTIS, 2006), we will use the position of features that are seen by both robots to find the transformations between their coordinate systems. A feature must be a SUMO object, as it needs to be something physical that exists in space/time to be perceived spatially by the robot. The existence of the same object in two different maps at the same time implies that the robots are in the same space. Since we are doing map merging in C-SLAM, we can at least assume that two robots are seeing one another and know it. To assume that a robot might know the location of another one when it sees it is not impractical, since this can be achieved by tagging each robot with a qr-code or using some wireless protocol for robot communication. Let's represent this state of affairs in the ontology:

First, let's declare that the robots and their coordinate systems exist:

$$\exists r1, c1 AutonomousRobot(r1) \wedge CS(c1) \wedge ref(c1, r1)$$

$$\exists r2, c2 AutonomousRobot(r2) \wedge CS(c2) \wedge ref(c2, r2)$$

If both robots are seeing one another, their position is represented in each others coordinate system. So, we have:

$$\exists p1, p2 PositionPoint(p1) \wedge PositionPoint(p2) \wedge in(p1, c1) \wedge in(p2, c2) \wedge pos(r1, p1) \wedge pos(r1, p2)$$

$$\exists p3, p4 PositionPoint(p3) \wedge PositionPoint(p4) \wedge in(p3, c1) \wedge in(p4, c2) \wedge pos(r2, p3) \wedge pos(r2, p4)$$

with $p1$ and $p4$ having values $(0, 0)$ since they are the center of their CartesianCoordinateSystems.

Since both robots are using the same orientation for the bases of their coordinate system, it is possible to move one to another using only a rotation for the change of bases followed by a translation. Let's show how the transformation from $c1$ to $c2$ can be constructed:

First, let's do the rotation by building a change of basis matrix that will transform the base vectors of $c1$ to $c2$. From linear algebra, we know that, for every point $\overrightarrow{[a]}$, we will have

$$\overrightarrow{[a]}_{B2} = C * \overrightarrow{[a]}_{B1}$$

where $\overrightarrow{[a]}_B$ is $\overrightarrow{[a]}$'s coordinates according to base B and C is the change of basis matrix.

In other words, considering only the rotations necessary for the change of coordinate system, we will have points described in $c1$'s base $B1$ and to get their equivalents in $c2$'s base $B2$ we must discover the change of matrix C so that, for every point $\overrightarrow{a}$ in base $B1$, we can do $C * \overrightarrow{[a]}_{B1}$ to get $\overrightarrow{[a]}_{B2}$.

As we have seen in section 3.1, every base's vector can be transformed to the other using a rotation, so CR represents the rotation matrix generated by the angle θ times the

original basis of $c2$, where θ represents the angle from the orientation vector of $c1$ to the orientation vector of $c2$ and the original basis of $c2$ is, as previously defined, $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$.

To find θ it is necessary to discover the OrientationPoint of $r2$ in $r1$'s AngularCS. This information can be obtained by combining the positioning information of both robots by using the angles from the center of their Ccs to the line of measurement. The OrientationPoint's value is

$$\theta = \pi + (\beta - \alpha)$$

where $\beta = \arctan 2(y(p2), x(p2))$ and $\alpha = \arctan 2(y(p3), x(p3))$. This relation is shown clearly in figure 3.5.

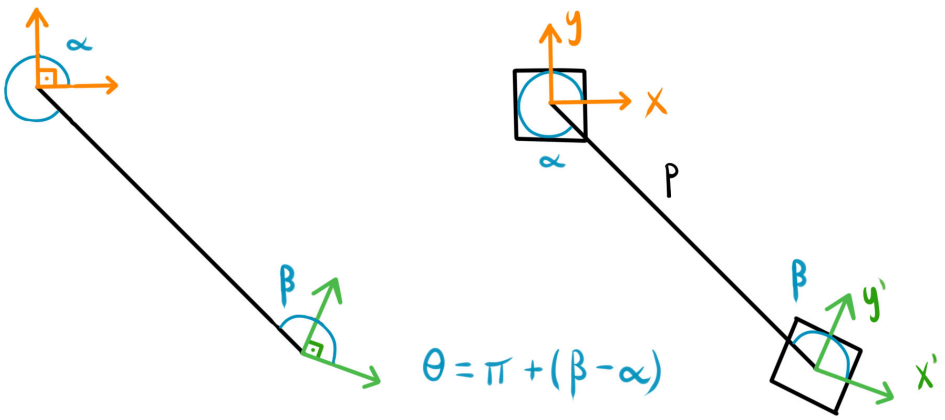

Figure 3.5: Obtaining θ, the angle of rotation that transforms one base to the other.

So, the rotation matrix is

$$RM = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

which leads us to the change of basis matrix C where

$$C = RM * \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} -\sin(\theta) & \cos(\theta) \\ \cos(\theta) & \sin(\theta) \end{bmatrix}$$

For the translation we have, trivially:

$$\begin{bmatrix} tx \\ ty \end{bmatrix} = \begin{bmatrix} x(p1) \\ y(p1) \end{bmatrix} - \begin{bmatrix} x(p4) \\ y(p4) \end{bmatrix}$$

and so, for a given point p in $c1$, the transformation to a point in $c2$ can be represented as:

$$T = \begin{bmatrix} -\sin(\theta) & \cos(\theta) \\ \cos(\theta) & \sin(\theta) \end{bmatrix} * \begin{bmatrix} x(p) \\ y(p) \end{bmatrix} + \begin{bmatrix} tx \\ ty \end{bmatrix}$$

In a real world scenario, determining the exact position of some object is no easy task due to measuring noise or general loss of precision. So, except from simulated environments, the exact position of an object will be determined with some confiability that is

determined appropriate by the application. This will make the merged map, too, correct only to some extent. This work does map merging in simulated, error-free scenarios, and so it does not address these problems. An approach that takes them into account using the same method of transformation can be found at (ZHOU; ROUMELIOTIS, 2006).

4 IMPLEMENTING THE EXTENSION

In this chapter we will present the actual implementation of our positioning primitives and how map merging and spatial reasoning was achieved.

4.0.1 Ontology representation

For the purposes of this work, we encoded CORA in OWL-DL, focusing on its positioning extension. Using OWL-DL is very convenient, as it is a popular standard with many modelling tools and reasoners available. This makes the ontology more user friendly and easy to port to other frameworks. As seen in section 2.2.2, OWL-DL is less expressive than SUO-Kif, but for this application it is expressive enough. When representing an ontology in some logic language, there is often a trade-off between the exprensiveness of said language and the speed and decidability of reasoning in it. In this case, we want to be able to apply map merging *during* C-SLAM, so we will focus on performance and use a reasoner specially designed for this application to materialize what we cannot infer from the ontology alone. More on how this was accomplished is present at section 4.0.5.

4.0.2 Representing CORA in OWL-DL

Since we chose OWL-DL as the language used to represent the ontology, we must represent CORA and its positioning extension using it. First, we will encode CORA in OWL. Since CORA uses SUMO as its top ontology, we will need to see this mapping in the ontology. SUMO has an OWL version, but using it gives us no advantage. In fact, since this version is composed mainly of things we will not need in the application (like all airports in the world), we might as well encode the few needed concepts of SUMO ourselves and add CORAs concepts to it.

4.0.3 Representing positioning in CORA in OWL-DL

We represented in OWL CORA as described in (PRESTES et al., 2013) and adding its positioning extension as defined in (J. L. CARBONERA S. R. FIORINI, 2013). With only this components, the concepts in the ontology are as shown in figure 4.1.

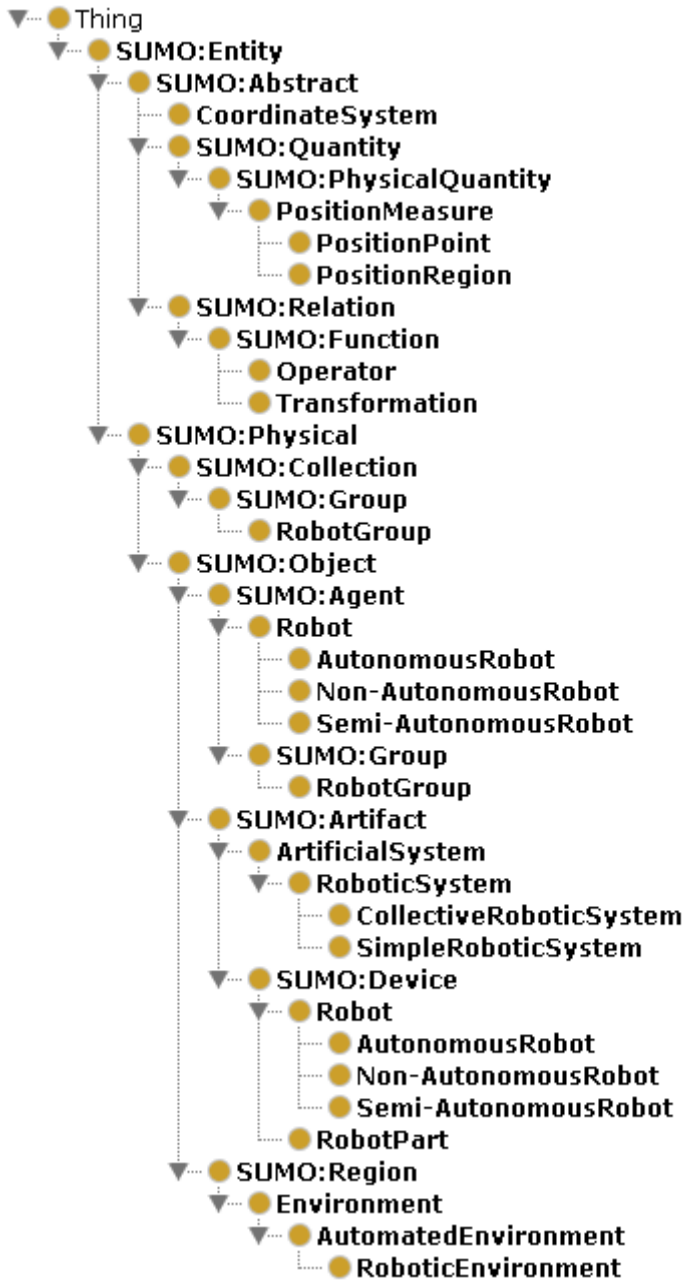

Figure 4.1: Taxonomy of CORAs concepts including the positioning extension. The SUMO prefix marks the concepts from SUMO.

4.0.4 Representing cartesian positioning

Since we will be using OWL-DL instead of SUO-KIF, we will need to represent our SUO-KIF cartesian primitives defined in section 3.1 in OWL-DL.

4.0.4.1 CartesianPositionPoint

A CartesianPositionPoint is a class that is subsumed by PositionPoint. Its coordinates $(x(p), y(p))$ are represented using datatype properties cartesianX and cartesianY, respectively. Using cardinality restrictions on those properties we can represent that every CartesianPositionPoint has to have *exactly one* X and Y coordinate.

Using a restriction on the object property inCS, that links a PositionPoint to its CoordinateSystem, we can say that a CartesianPositionPoint is *only* in a CartesianCoordinateSystem.

4.0.4.2 CartesianCoordinateSystem

A CCs will be a class that is subsumed by CoordinateSystem. Using a restriction on the object property ofCS, that is, the inverse object property of inCS, we can say that a CCs has *only* CartesianPositionPoints in it.

Restringing the coordinates of the reference object of the CCs to $(0, 0)$ is the same as restricting the ref object property, that links a Ccs to its reference object, to only objects that have a position $(0, 0)$ in this CCs. While it is possible to restrict an object property to have as its range only points with coordinates of some given value, OWL is not so expressive that we can restrict ref(cs,o) to "o is an objects with position $(0, 0)$ in cs". The problem is that in OWL we cannot reuse variables, so we are not able to restrict ref(cs,o) if we need to use the variable cs inside the restrictions.

This means that this restriction will have to be encoded in the application. In our case this is easy, as the position a robot receives when it starts C-SLAM is $(0, 0)$ in the CCs that has said robot as its reference object.

4.0.4.3 CartesianPositionRegions

CartesianPositionRegions will be the regions in our Cartesian space. This means that all points inside the region must be CartesianPositionPoints. This can be achieved creating an object property that is the inverse of inPR to represent the points of a PositionRegion and restricting it to contain *only* CartesianPositionPoints.

4.0.4.4 CartesianOperators

One of the common problems with OWL is the lack of ternary relations. This is the case of the Operator, that is relation between two objects and also generates a region. Following a very popular OWL desing pattern, we will represent an Operator as an OWL class and, for each member of the ternary relation, add a new object property from the Operator class to it. This means that our Operator class will have three object properties associated to it, one representing the domain, other the range and a third the region the operator generates.

Since a CartesianOperator is one that generates CartesianPositionRegions, it will be a subclass of Operator that has a restriction on the object property that represents the generated region so that the operator will *only* generate CartesianPositionRegions.

4.0.4.5 AngularOrientationPoint and AngularOrientationCoordinateSystem

Since they were left out of the SUO-KIF version, we will leave them out of the OWL one.

4.0.5 Software architecture

This description of the ontology does not represent or apply the transformations and operators described in chapter 3.1. For this kind of reasoning, simple DL reasoning is not enough. This means one must build or use some rule system that will materialize the operators and transformations of the ontology. What's more, this system must be extensible enough so that it can support any new coordinate system that might be developed in the future alongside the one described here. It must also be fast enough that the reasoning can run in real time while the robot is exploring the ambient. While there are many rule systems for OWL-DL, such as SWRL(IAN HORROCKS PETER F. PATEL-SCHNEIDER, 2013), but we are looking for a definition of reasoning that is not linked to any specific language. In this work an extensible framework for reasoning and representing spatial information was developed in Java.

The framework represents the positioning primitives of CORA as abstract classes and does reasoning over them. Using reflection, the system allows the user to register new classes to represent primitives extending the ones in the positioning extension for CORA. All the user needs to do is create the extended primitives in the ontology and provide their mapping to Java classes.

The main architecture is, as follows:

4.0.5.1 Ontology Model

A model that represents the ontology in OWL-DL. The JENA(FOUNDATION, 2013) library was used to this purpose.

4.0.5.2 Spatial Mapper

This component is where classes are registered so that the reasoner can get the correct Java class for each OWL one. Classes can be registered during runtime.

4.0.5.3 Spatial Reasoner

Performs the reasoning using only the abstract classes that represent the positioning primitives of CORAPOS.

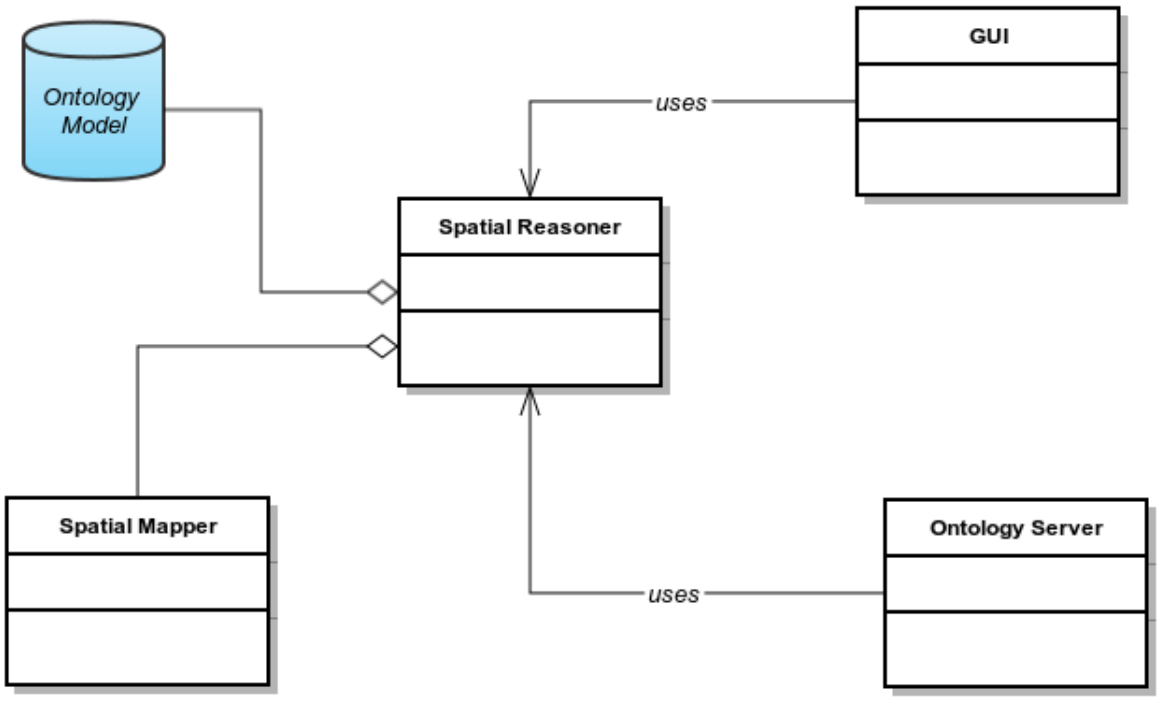

Figure 4.2: Main components of the framework

4.0.5.4 Ontology Server

A multi-threaded server that connects the robots and the application using TCP sockets. Its purpose is to receive the spatial information sensed by the robots and return the inferred information.

4.0.5.5 GUI

Shows the robots coordinate systems with their respective PositionPoints.

4.0.5.6 Spatial Reasoning

When the reasoner starts, it goes through four stages:

1. The reasoner is started receiving a spatial mapper and a connection to the ontology.
2. The reasoner finds which OWL-DL classes represent spatial information in the given ontology and asks for their equivalent Java versions to the spatial mapper.
3. All the classes found are intialized and kept in the reasoner.
4. The reasoner tries to form transformations between the existing coordinate systems using the transformations available in the spatial mapper. If any is found, a map merge is performed.
5. The reasoner applies the operators provided by the spatial mapper to the existing

objects to obtain position regions. If any object happens to reside in those regions, the operator is instantiated.

When the robot perceives some object and finds its PositionPoint, the reasoner will receive this new PositionPoint and do:

1. Use the existing transformations to generate its equivalent points in other CoordinateSystems.
2. Try to see if any new transformation can be formed.
3. Try to form operators using the new PositionPoints and the existing ones.

This saves a lot of processing time as the reasoner keeps tracks of inferred points and transformations. The use of Java classes also makes the spatial representations more efficient as it allows for the mathematically intense part of transformations and operators to be done more directly. For example, consider the operator "near" that generates the PositionRegion near an object $o1$. If we use a Java class to represent the region, it can determine if an object $o2$ is inside it using only the distance from $o1$ to $o2$ instead of having to represent the region as a generic polygon region in the ontology.

5 EXPERIMENTAL EVALUATION

In this chapter we will present some of the results obtained by static map merging using the framework.

5.1 Experiment Description

For our experiment, we will use MobileSimulator(COMMUNITY, 2013) with the Aria library to represent static map merging with two pioneer robots performing C-SLAM. The robots start at a position determined by the ontology where they can see each other. Since each robot is seeing each other, the reasoner will find a transformation from their CartesianCoordinateSystems.

Using their sonar sensors, both robots acquire spatial information in the form of CartesianPositionPoints. Following the procedure described in 4.0.5.6, the maps are merged as the spatial information is added to the ontology.

The Cartesian primitives described in section 4 were implemented in Java and their mapping was registered at the start of the experiment. Transformations following the procedures described in 3.3 were also implemented in Java and registered so that the map merging could be done automatically for any two CartesianCoordinateSystems upon robot encounter.

5.2 Results

A figure of the merged maps in the beginning of the process can be seeing at figure 5.1. As the process begins, more points are added to the ontology and merged. This can be seen in figure 5.2.

5.3 Evaluation

This is still a very simple example as it is map merging in a static context. On the other hand, it shows that this architecture can be used to solve this problem. Since the ontology used in this example is synchronic, it does not allow for changes in position. In the next chapter we will discuss how this can be achieved.

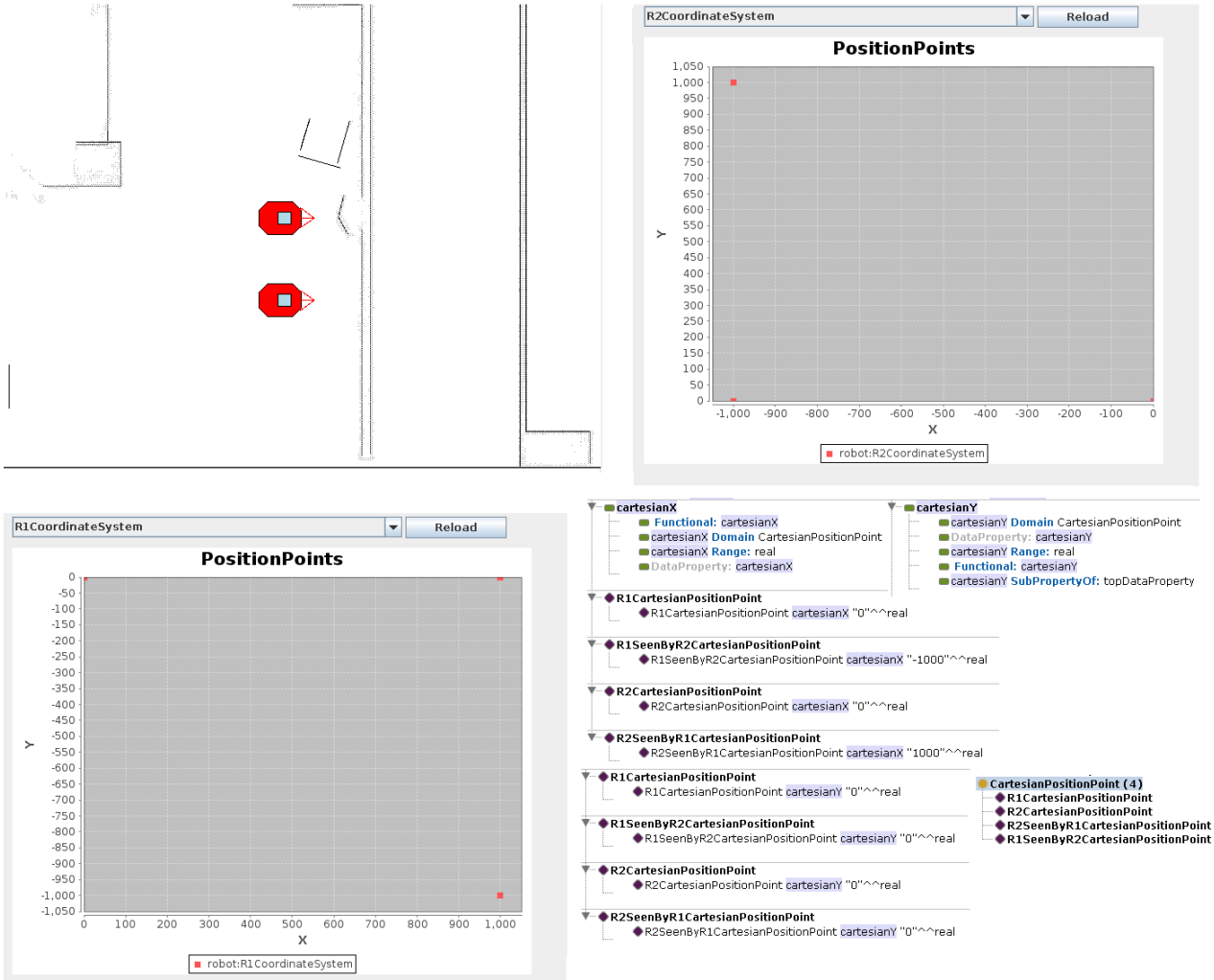

Figure 5.1: State at the start of C-SLAM.

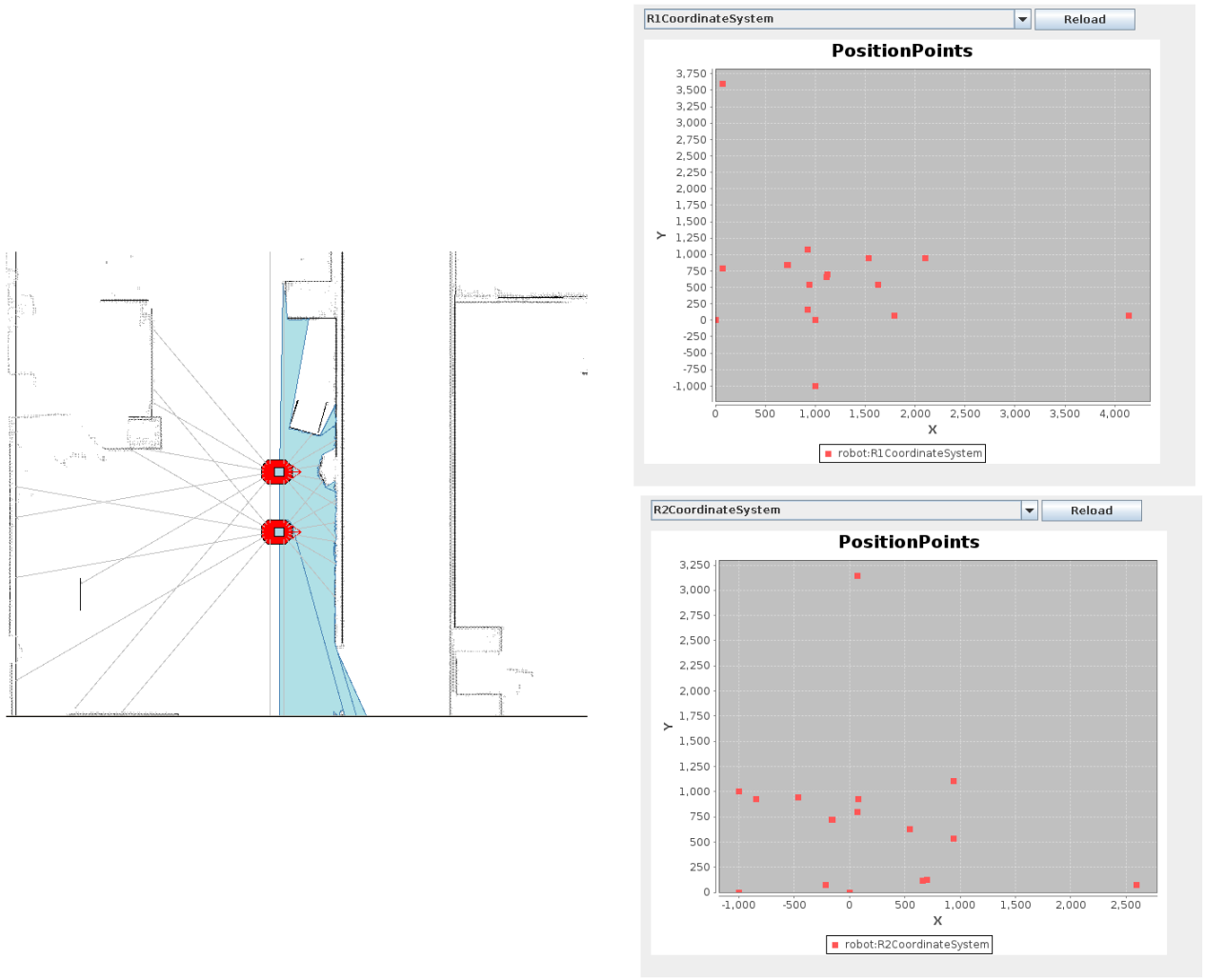

Figure 5.2: State after the initial merge of both maps.

6 DIACHRONALIZING CORA WITH STATIC POSITIONING

In this chapter we will show the limitations of the current state of positioning in CORA has for the domain of SLAM. We will also introduce the changes needed to surpass this limitations.

SLAM is a process that happens over time. So, considering a robot that is performing SLAM in a room, representing the spatial information it gathered in a synchronic way does not represent the whole process.

A diachronic ontology is one that can represent the change of state of its concepts through time. SUMO already has support for indexing facts over time. In fact, SUMO represents time using TimeMeasures that are positions or intervals in the universal timeline that goes from negative to positive infinity. To represent that an object is located somewhere in a given time, one uses (locatedAtTime ?obj ?time ?place), that in SUMO is equivalent to (holdsDuring ?time (located ?obj ?place))).

So, using the same principles applied in SUMO, we can diachronalize positioning in CORA by using temporal relations. Instead of saying that an object has some position measure, we will say that it has that position measure at that time using the relation posAtTime. It is trivial to see that this applies to many other relations in CORA itself, from being part of a RobotGroup to having a certain RobotPart.

6.0.1 Coordinate System

An interesting case is the one regarding the coordinate system. Coordinate systems are defined in relation to some reference object. Since a coordinate system is a spatial primitive, it is safe to say that it is defined in relation to the position/orientation of its reference object. In a synchronic ontology this position/orientation never changes, so this definition is fine, but as soon as we think about the robot moving and rotating, we need to ask some fundamental questions. Considering the coordinate system as originally designed, as the robot moves, does its coordinate system stay the same or does any movement provide a new one? If we look for the definition of coordinate system in (J. L. CARBONERA S. R. FIORINI, 2013), we will see that it must be homeomorphic to $\mathbb{R}^n$. This means that we cannot reuse the old definition of coordinate system where the coordinate system is directly linked to the reference object, else two positions with the same value will be mapped to different points in $\mathbb{R}^n$ and the homeomorphic restriction won't be met.

Giving a more practical example in a $2D$ space, consider the situation described in

figure 6.1, where the robot is moving in circles near two green balls positioned symmetrically. Using the old definition of coordinate system, when the robot senses the second ball, its position will be the same as the one assigned to the first one and so the coordinate system would not be homeomorphic to $\mathbb{R}^2$.

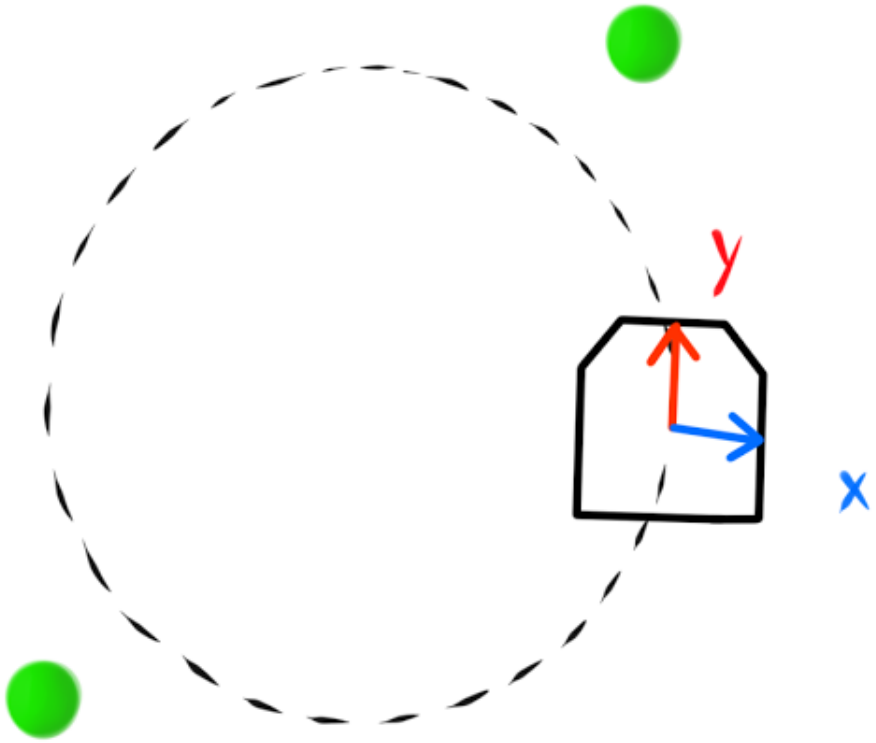

Figure 6.1: A circular robot trajectory leading to positioning inconsistencies in the old definition of coordinate system.

It is better, then, for a coordinate system to have as its reference the position of an object *at* some given time. To be exact, we will go from:

(ref ?cs ?object)

to

(ref ?cs ?object ?time)

where

```
(=>
        (ref ?cs ?object ?time)
        (holdsDuring ?time
                (exactlyLocated ?object ?location)
        )
)
```

This notion of a coordinate system being defined by the location of an object at some time is an intuitive notion, as it is easy to visualize a robot setting its coordinate system based on its position when turned on and continuing to use the same coordinate system as it explores a room.

It also keeps transformations between coordinate systems timeless, which is very useful as they only need to be calculated once. Unfortunately, a robot exploring some space that wants to always use the same coordinate system in the ontology will still have to do the translation from said coordinate system to the one it has locally, but there is no escape from this translation. What we have now is the *choice* of whether or not this transformation is represented explicitly in the ontology.

6.0.2 PositionMeasure versus PositionRepresentation

A PositionMeasure in the spatial extension to CORA is the measure of some object's position. This means that, if we were to see a robot move in circles, every time it reached the start of the circle determined by the point po it would not say that its position is po. Instead, it would say that its position is another PositionMeasure po' that has the same coordinates as po. This view defines that the identity of PositionMeasure is the *measurement* of the position of some object and not its abstract, error-free position. This view implies that many "duplicates" will be generated as the robot is always sensing its environment and closing loops.

Representing only the measured position also means that, if one wanted to define the robot's goal as some arbitrary point in an unexplored part of the ambient, this point would not be a measure of the position of any object and therefore cannot be represented in the ontology.

For this two reasons, henceforth we will change PositionMeasures to PositionRepresentations in the ontology, taking away from the identity of a PositionMeasure the necessity of having sensed an object in said position.

Figure 6.2 shows the change in the taxonomy along with the new relation posAtTime.

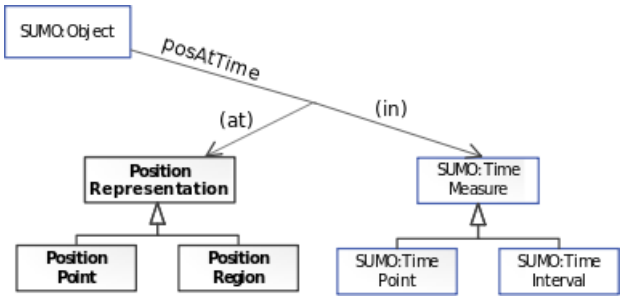

Figure 6.2: Every object is located at some PositionRepresentation in a certain time.

6.0.3 Operators

Since an object that is in some region might leave it afterwards, the notion of operator must also take time into account. Once again, we will simply add another argument representing that the operator holds at some time. So, while earlier we said (near ?obj1

?obj2), now we say (near ?obj1 ?obj2 ?time).

6.0.4 Conclusion

Extending an ontology helps one understand its ontological commitments, but one should never forget that an ontology defines the important concepts in the domain and so, when adding new features, it is of utmost importance to respect the old definitions only to extent that they are still representing correctly the concepts of the domain as seen by the community.

6.0.5 Ontology representation

To diachronalize the ontology, ternary relations were used. In order to represent them in OWL-DL, one needs to turn the relation itself into a concept and connect it to its arguments using one property for each. For example, in (locatedAtTime ?obj ?time ?place), locatedAtTime will become a concept of type TernaryPredicate and properties locatedAtTimeObject, locatedAtTimeTime and locatedAtTimePlace will connect it to its arguments.

6.0.6 Software architecture and spatial reasoning

The software architecture and spatial resoning do not differ so much between the synchronic and diachronic versions. In fact, the same architecture can be used provided the user implementations for the operators takes time into account. However, this scenario could not be tested in time and was left for future work.

7 CONCLUSIONS AND FUTURE WORK

We have defined and implemented an extension for the positioning extension for CORA. Using it we defined positioning and map merging for land robots performing C-SLAM. A simulation was executed for the synchronic version and we also laid the basis for a diachronic version of the ontology. We have shown how qualitative and quantitative spatial information can be represented and how spatial reasoning can be performed. With this ontological approach we have demonstrated how ontologies can be used to represent and reason over spatial data, providing the means for precise communication and thus making cooperation possible.

7.1 Future Work

In the future, the diachronic version of the ontology can be used in a C-SLAM application where multiple robots move in real time. Other types of coordinate system could be added to the application in order to provide scenarios where different types of robots cooperate. An aerial robot, for example, could cooperate with a land robot to find the environments features. Furthermore, sensors in the environment could also participate in SLAM using the ontology. For example, the information obtained by motion sensors in a house might help locate the robot.

REFERENCES

ALLEN, J. F.; HAYES, P. J. A Common-sense Theory of Time. In: INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE - VOLUME 1, 9., San Francisco, CA, USA. **Proceedings...** Morgan Kaufmann Publishers Inc., 1985. p.528–531. (IJCAI'85).

ARNDT, M. et al. Performance evaluation of ambient services by combining robotic frameworks and a smart environment platform. **Robotics and Autonomous Systems**, [S.l.], v.61, n.11, p.1173 – 1185, 2013. <ce:title>Ubiquitous Robotics</ce:title>.

ASHBURNER, M. et al. Gene ontology: tool for the unification of biology. the gene ontology consortium. **Nature genetics**, Department of Genetics, Stanford University School of Medicine, California, USA. cherry@stanford.edu, v.25, n.1, p.25–29, May 2000.

BATEMAN, J.; FARRAR, S. **Spatial Ontology Baseline**. 2006.

CARPIN, S.; BIRK, A.; JUCIKAS, V. On map merging. **Robotics and Autonomous Systems**, [S.l.], v.53, n.1, p.1 – 14, 2005.

COMMUNITY, A. M. http://robots.mobilerobots.com/wiki/MobileSim, 2013.

CRAIG SCHLENOFF EDSON PRESTES, P. G. J. L. C. S. R. F. V. A. M. J. M. A. P. J. S. G. **Draft Standard for Ontologies for Robotics and Automation**. 2013.

FOUNDATION, A. S. http://jena.apache.org/, 2013.

GANGEMI, A. et al. Sweetening Ontologies with DOLCE. In: GóMEZ-PéREZ, A.; BENJAMINS, V. (Ed.). **Knowledge Engineering and Knowledge Management**: ontologies and the semantic web. [S.l.]: Springer Berlin Heidelberg, 2002. p.166–181. (Lecture Notes in Computer Science, v.2473).

GROUP, S. U. O. W. http://suo.ieee.org/SUO/KIF/suo-kif.html, 2013.

GROUP, S. U. O. W. http://suo.ieee.org/SUO/KIF/, 2013.

GROUP, S. U. O. W. http://www.ontologyportal.org/SUMO.owl, 2013.

IAN HORROCKS PETER F. PATEL-SCHNEIDER, H. B. S. T. B. G. M. D. http://www.w3.org/Submission/SWRL/, 2013.

J. L. CARBONERA S. R. FIORINI, E. P. V. A. M. J. M. A. R. M. A. L. P. G. T. H. M. E. B. e. C. S. Defining Position in a Core Ontology for Robotics. In: IEEE/RSJ INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS (IROS 2013), TOKYO, JAPAN. **Proceedings...** [S.l.: s.n.], 2013.

KLEER, J. de; BROWN, J. S. A Qualitative Physics Based on Confluences. In: BOBROW, D. G. (Ed.). **Qualitative Reasoning about Physical Systems**. Cambridge, MA: MIT Press, 1985. p.7–83.

KNUBLAUCH, H. et al. The Protégé OWL plugin: an open development environment for semantic web applications. In: **Anais...** Springer, 2004. p.229–243.

MICHAEL K. SMITH, C. W.; DEBORAH L. MCGUINNESS, E. **OWL Web Ontology Language Guide**. http://www.w3.org/TR/2004/REC-owl-guide-20040210/, 2004.

NILES, I.; PEASE, A. Towards a Standard Upper Ontology. In: INTERNATIONAL CONFERENCE ON FORMAL ONTOLOGY IN INFORMATION SYSTEMS - VOLUME 2001, New York, NY, USA. **Proceedings...** ACM, 2001. p.2–9. (FOIS '01).

PRESTES, E. et al. Towards a core ontology for robotics and automation. **Robotics and Autonomous Systems**, [S.l.], v.61, n.11, p.1193 – 1204, 2013. <ce:title>Ubiquitous Robotics</ce:title>.

SCHLENOFF, C. et al. An IEEE standard Ontology for Robotics and Automation. In: INTELLIGENT ROBOTS AND SYSTEMS (IROS), 2012 IEEE/RSJ INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2012. p.1337–1342.

SIRIN, E. et al. Pellet: a practical owl-dl reasoner. **Web Semantics: Science, Services and Agents on the World Wide Web**, [S.l.], v.5, n.2, p.51 – 53, 2007. <ce:title>Software Engineering and the Semantic Web</ce:title>.

SOWA, J. F. Top-level ontological categories. **Int. J. Hum.-Comput. Stud.**, [S.l.], v.43, n.5-6, p.669–685, 1995.

STUDER, R.; BENJAMINS, V.; FENSEL, D. Knowledge engineering: principles and methods. **Data & Knowledge Engineering**, [S.l.], v.25, n.1–2, p.161 – 197, 1998.

ZHOU, X.; ROUMELIOTIS, S. Multi-robot SLAM with Unknown Initial Correspondence: the robot rendezvous case. In: INTELLIGENT ROBOTS AND SYSTEMS, 2006 IEEE/RSJ INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2006. p.1785–1792.

AppendixA

A.1 OWL-DL CORA with Cartesian Positioning

The ontology:

A.1.1 Ontology

```
<?xml version="1.0"?>


<!DOCTYPE rdf:RDF [
    <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
    <!ENTITY RobotsAutomation "http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#" >
    <!ENTITY SUMO "http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#SUMO:" >
]>


<rdf:RDF xmlns="http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#"
     xml:base="http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl"
     xmlns:RobotsAutomation="http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:SUMO="&RobotsAutomation;SUMO:">
    <owl:Ontology rdf:about="http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl"/>



    <!--
    ////////////////////////////////////////////////////////////////////////////////////
    //
    // Object Properties
    //
    ////////////////////////////////////////////////////////////////////////////////////
     -->




    <!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#SUMO:part -->

    <owl:ObjectProperty rdf:about="&RobotsAutomation;SUMO:part">
        <rdf:type rdf:resource="&owl;AsymmetricProperty"/>
        <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    </owl:ObjectProperty>



    <!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#SUMO:properPart -->

    <owl:ObjectProperty rdf:about="&RobotsAutomation;SUMO:properPart">
        <rdf:type rdf:resource="&owl;AsymmetricProperty"/>
        <rdf:type rdf:resource="&owl;TransitiveProperty"/>
        <rdfs:subPropertyOf rdf:resource="&RobotsAutomation;SUMO:part"/>
    </owl:ObjectProperty>



    <!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#equippedWith -->

    <owl:ObjectProperty rdf:about="&RobotsAutomation;equippedWith">
        <rdfs:domain rdf:resource="&RobotsAutomation;RoboticEnvironment"/>
        <rdfs:range rdf:resource="&RobotsAutomation;RoboticSystem"/>
    </owl:ObjectProperty>
```

```
<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#in -->

<owl:ObjectProperty rdf:about="&RobotsAutomation;in">
    <rdf:type rdf:resource="&owl;FunctionalProperty"/>
    <rdfs:range rdf:resource="&RobotsAutomation;CoordinateSystem"/>
    <rdfs:domain rdf:resource="&RobotsAutomation;PositionPoint"/>
</owl:ObjectProperty>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#inPR -->

<owl:ObjectProperty rdf:about="&RobotsAutomation;inPR">
    <rdfs:domain rdf:resource="&RobotsAutomation;PositionPoint"/>
    <rdfs:range rdf:resource="&RobotsAutomation;PositionRegion"/>
</owl:ObjectProperty>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#memberOf -->

<owl:ObjectProperty rdf:about="&RobotsAutomation;memberOf">
    <rdfs:domain rdf:resource="&RobotsAutomation;Robot"/>
    <rdfs:range rdf:resource="&RobotsAutomation;RobotGroup"/>
</owl:ObjectProperty>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#ofCS -->

<owl:ObjectProperty rdf:about="&RobotsAutomation;ofCS">
    <rdf:type rdf:resource="&owl;InverseFunctionalProperty"/>
    <rdfs:domain rdf:resource="&RobotsAutomation;CoordinateSystem"/>
    <rdfs:range rdf:resource="&RobotsAutomation;PositionPoint"/>
    <owl:inverseOf rdf:resource="&RobotsAutomation;in"/>
</owl:ObjectProperty>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#operatorDomain -->

<owl:ObjectProperty rdf:about="&RobotsAutomation;operatorDomain">
    <rdfs:domain rdf:resource="&RobotsAutomation;Operator"/>
    <rdfs:range rdf:resource="&RobotsAutomation;SUMO:Object"/>
</owl:ObjectProperty>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#operatorGenerates -->

<owl:ObjectProperty rdf:about="&RobotsAutomation;operatorGenerates">
    <rdfs:domain rdf:resource="&RobotsAutomation;Operator"/>
    <rdfs:range rdf:resource="&RobotsAutomation;PositionRegion"/>
</owl:ObjectProperty>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#operatorRange -->

<owl:ObjectProperty rdf:about="&RobotsAutomation;operatorRange">
    <rdfs:domain rdf:resource="&RobotsAutomation;Operator"/>
    <rdfs:range rdf:resource="&RobotsAutomation;SUMO:Object"/>
</owl:ObjectProperty>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#pos -->

<owl:ObjectProperty rdf:about="&RobotsAutomation;pos">
    <rdfs:range rdf:resource="&RobotsAutomation;PositionMeasure"/>
    <rdfs:domain rdf:resource="&RobotsAutomation;SUMO:Object"/>
</owl:ObjectProperty>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#ptsOfPR -->

<owl:ObjectProperty rdf:about="&RobotsAutomation;ptsOfPR">
    <rdfs:range rdf:resource="&RobotsAutomation;PositionPoint"/>
    <rdfs:domain rdf:resource="&RobotsAutomation;PositionRegion"/>
    <owl:inverseOf rdf:resource="&RobotsAutomation;inPR"/>
</owl:ObjectProperty>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#ref -->

<owl:ObjectProperty rdf:about="&RobotsAutomation;ref">
    <rdf:type rdf:resource="&owl;FunctionalProperty"/>
    <rdfs:domain rdf:resource="&RobotsAutomation;CoordinateSystem"/>
    <rdfs:range rdf:resource="&RobotsAutomation;SUMO:Object"/>
</owl:ObjectProperty>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#transformationMapsFrom -->
```

```
<owl:ObjectProperty rdf:about="&RobotsAutomation;transformationMapsFrom">
    <rdfs:domain rdf:resource="&RobotsAutomation;Transformation"/>
    <rdfs:range>
        <owl:Restriction>
            <owl:onProperty rdf:resource="&RobotsAutomation;transformationMapsFrom"/>
            <owl:onClass rdf:resource="&RobotsAutomation;CoordinateSystem"/>
            <owl:qualifiedCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
        </owl:Restriction>
    </rdfs:range>
</owl:ObjectProperty>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#transformationMapsTo -->

<owl:ObjectProperty rdf:about="&RobotsAutomation;transformationMapsTo">
    <rdfs:range rdf:resource="&RobotsAutomation;CoordinateSystem"/>
    <rdfs:domain rdf:resource="&RobotsAutomation;Transformation"/>
</owl:ObjectProperty>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#workIn -->

<owl:ObjectProperty rdf:about="&RobotsAutomation;workIn">
    <rdfs:domain rdf:resource="&RobotsAutomation;Robot"/>
    <rdfs:range rdf:resource="&RobotsAutomation;RobotGroup"/>
</owl:ObjectProperty>



<!--
///////////////////////////////////////////////////////////////////////////////////////
//
// Data properties
//
///////////////////////////////////////////////////////////////////////////////////////
 -->



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#cartesianX -->

<owl:DatatypeProperty rdf:about="&RobotsAutomation;cartesianX">
    <rdf:type rdf:resource="&owl;FunctionalProperty"/>
    <rdfs:domain rdf:resource="&RobotsAutomation;CartesianPositionPoint"/>
    <rdfs:range rdf:resource="&owl;real"/>
</owl:DatatypeProperty>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#cartesianY -->

<owl:DatatypeProperty rdf:about="&RobotsAutomation;cartesianY">
    <rdf:type rdf:resource="&owl;FunctionalProperty"/>
    <rdfs:domain rdf:resource="&RobotsAutomation;CartesianPositionPoint"/>
    <rdfs:range rdf:resource="&owl;real"/>
    <rdfs:subPropertyOf rdf:resource="&owl;topDataProperty"/>
</owl:DatatypeProperty>



<!--
///////////////////////////////////////////////////////////////////////////////////////
//
// Classes
//
///////////////////////////////////////////////////////////////////////////////////////
 -->



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#ArtificialSystem -->

<owl:Class rdf:about="&RobotsAutomation;ArtificialSystem">
    <rdfs:subClassOf rdf:resource="&RobotsAutomation;SUMO:Artifact"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#AutomatedEnvironment -->

<owl:Class rdf:about="&RobotsAutomation;AutomatedEnvironment">
    <rdfs:subClassOf rdf:resource="&RobotsAutomation;Environment"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#AutonomousRobot -->

<owl:Class rdf:about="&RobotsAutomation;AutonomousRobot">
    <rdfs:subClassOf rdf:resource="&RobotsAutomation;Robot"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#CartesianBackOf -->
```

```
<owl:Class rdf:about="&RobotsAutomation;CartesianBackOf">
    <rdfs:subClassOf rdf:resource="&RobotsAutomation;CartesianOperator"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#CartesianCoordinateSystem -->

<owl:Class rdf:about="&RobotsAutomation;CartesianCoordinateSystem">
    <rdfs:subClassOf rdf:resource="&RobotsAutomation;CoordinateSystem"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="&RobotsAutomation;ofCS"/>
            <owl:allValuesFrom rdf:resource="&RobotsAutomation;CartesianPositionPoint"/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#CartesianFrontOf -->

<owl:Class rdf:about="&RobotsAutomation;CartesianFrontOf">
    <rdfs:subClassOf rdf:resource="&RobotsAutomation;CartesianOperator"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#CartesianLeftOf -->

<owl:Class rdf:about="&RobotsAutomation;CartesianLeftOf">
    <rdfs:subClassOf rdf:resource="&RobotsAutomation;CartesianOperator"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#CartesianOperator -->

<owl:Class rdf:about="&RobotsAutomation;CartesianOperator">
    <rdfs:subClassOf rdf:resource="&RobotsAutomation;Operator"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="&RobotsAutomation;operatorGenerates"/>
            <owl:allValuesFrom rdf:resource="&RobotsAutomation;CartesianPositionRegion"/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#CartesianPositionPoint -->

<owl:Class rdf:about="&RobotsAutomation;CartesianPositionPoint">
    <rdfs:subClassOf rdf:resource="&RobotsAutomation;PositionPoint"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="&RobotsAutomation;cartesianX"/>
            <owl:qualifiedCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
            <owl:onDataRange rdf:resource="&owl;real"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="&RobotsAutomation;cartesianY"/>
            <owl:qualifiedCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
            <owl:onDataRange rdf:resource="&owl;real"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="&RobotsAutomation;in"/>
            <owl:allValuesFrom rdf:resource="&RobotsAutomation;CartesianCoordinateSystem"/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#CartesianPositionRegion -->

<owl:Class rdf:about="&RobotsAutomation;CartesianPositionRegion">
    <rdfs:subClassOf rdf:resource="&RobotsAutomation;PositionRegion"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="&RobotsAutomation;ptsOfPR"/>
            <owl:allValuesFrom rdf:resource="&RobotsAutomation;CartesianPositionPoint"/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#CartesianRightOf -->

<owl:Class rdf:about="&RobotsAutomation;CartesianRightOf">
    <rdfs:subClassOf rdf:resource="&RobotsAutomation;CartesianOperator"/>
</owl:Class>
```

```
<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#CollectiveRoboticSystem -->

<owl:Class rdf:about="&RobotsAutomation;CollectiveRoboticSystem">
    <rdfs:subClassOf rdf:resource="&RobotsAutomation;RoboticSystem"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="&RobotsAutomation;SUMO:properPart"/>
            <owl:onClass rdf:resource="&RobotsAutomation;RobotGroup"/>
            <owl:qualifiedCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#CoordinateSystem -->

<owl:Class rdf:about="&RobotsAutomation;CoordinateSystem">
    <rdfs:subClassOf rdf:resource="&RobotsAutomation;SUMO:Abstract"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#Environment -->

<owl:Class rdf:about="&RobotsAutomation;Environment">
    <rdfs:subClassOf rdf:resource="&RobotsAutomation;SUMO:Region"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#Non-AutonomousRobot -->

<owl:Class rdf:about="&RobotsAutomation;Non-AutonomousRobot">
    <rdfs:subClassOf rdf:resource="&RobotsAutomation;Robot"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#Operator -->

<owl:Class rdf:about="&RobotsAutomation;Operator">
    <rdfs:subClassOf rdf:resource="&RobotsAutomation;SUMO:Function"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#PositionMeasure -->

<owl:Class rdf:about="&RobotsAutomation;PositionMeasure">
    <rdfs:subClassOf rdf:resource="&RobotsAutomation;SUMO:PhysicalQuantity"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#PositionPoint -->

<owl:Class rdf:about="&RobotsAutomation;PositionPoint">
    <rdfs:subClassOf rdf:resource="&RobotsAutomation;PositionMeasure"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="&RobotsAutomation;in"/>
            <owl:onClass rdf:resource="&RobotsAutomation;CoordinateSystem"/>
            <owl:qualifiedCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#PositionRegion -->

<owl:Class rdf:about="&RobotsAutomation;PositionRegion">
    <rdfs:subClassOf rdf:resource="&RobotsAutomation;PositionMeasure"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#Robot -->

<owl:Class rdf:about="&RobotsAutomation;Robot">
    <rdfs:subClassOf>
        <owl:Class>
            <owl:intersectionOf rdf:parseType="Collection">
                <rdf:Description rdf:about="&RobotsAutomation;SUMO:Agent"/>
                <rdf:Description rdf:about="&RobotsAutomation;SUMO:Device"/>
            </owl:intersectionOf>
        </owl:Class>
    </rdfs:subClassOf>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#RobotGroup -->

<owl:Class rdf:about="&RobotsAutomation;RobotGroup">
    <rdfs:subClassOf rdf:resource="&RobotsAutomation;SUMO:Group"/>
```

```
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#RobotPart -->

<owl:Class rdf:about="&RobotsAutomation;RobotPart">
    <rdfs:subClassOf rdf:resource="&RobotsAutomation;SUMO:Device"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="&RobotsAutomation;SUMO:properPart"/>
            <owl:someValuesFrom rdf:resource="&RobotsAutomation;Robot"/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#RoboticEnvironment -->

<owl:Class rdf:about="&RobotsAutomation;RoboticEnvironment">
    <rdfs:subClassOf rdf:resource="&RobotsAutomation;AutomatedEnvironment"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#RoboticSystem -->

<owl:Class rdf:about="&RobotsAutomation;RoboticSystem">
    <rdfs:subClassOf rdf:resource="&RobotsAutomation;ArtificialSystem"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#SUMO:Abstract -->

<owl:Class rdf:about="&RobotsAutomation;SUMO:Abstract">
    <rdfs:subClassOf rdf:resource="&RobotsAutomation;SUMO:Entity"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#SUMO:Agent -->

<owl:Class rdf:about="&RobotsAutomation;SUMO:Agent">
    <rdfs:subClassOf rdf:resource="&RobotsAutomation;SUMO:Object"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#SUMO:Artifact -->

<owl:Class rdf:about="&RobotsAutomation;SUMO:Artifact">
    <rdfs:subClassOf rdf:resource="&RobotsAutomation;SUMO:Object"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#SUMO:Collection -->

<owl:Class rdf:about="&RobotsAutomation;SUMO:Collection">
    <rdfs:subClassOf rdf:resource="&RobotsAutomation;SUMO:Physical"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#SUMO:Device -->

<owl:Class rdf:about="&RobotsAutomation;SUMO:Device">
    <rdfs:subClassOf rdf:resource="&RobotsAutomation;SUMO:Artifact"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#SUMO:Entity -->

<owl:Class rdf:about="&RobotsAutomation;SUMO:Entity"/>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#SUMO:Function -->

<owl:Class rdf:about="&RobotsAutomation;SUMO:Function">
    <rdfs:subClassOf rdf:resource="&RobotsAutomation;SUMO:Relation"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#SUMO:Group -->

<owl:Class rdf:about="&RobotsAutomation;SUMO:Group">
    <rdfs:subClassOf>
        <owl:Class>
            <owl:intersectionOf rdf:parseType="Collection">
                <rdf:Description rdf:about="&RobotsAutomation;SUMO:Agent"/>
                <rdf:Description rdf:about="&RobotsAutomation;SUMO:Collection"/>
            </owl:intersectionOf>
        </owl:Class>
    </rdfs:subClassOf>
```

```
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#SUMO:Object -->

<owl:Class rdf:about="&RobotsAutomation;SUMO:Object">
    <rdfs:subClassOf rdf:resource="&RobotsAutomation;SUMO:Physical"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#SUMO:Physical -->

<owl:Class rdf:about="&RobotsAutomation;SUMO:Physical">
    <rdfs:subClassOf rdf:resource="&RobotsAutomation;SUMO:Entity"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#SUMO:PhysicalQuantity -->

<owl:Class rdf:about="&RobotsAutomation;SUMO:PhysicalQuantity">
    <rdfs:subClassOf rdf:resource="&RobotsAutomation;SUMO:Quantity"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#SUMO:Quantity -->

<owl:Class rdf:about="&RobotsAutomation;SUMO:Quantity">
    <rdfs:subClassOf rdf:resource="&RobotsAutomation;SUMO:Abstract"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#SUMO:Region -->

<owl:Class rdf:about="&RobotsAutomation;SUMO:Region">
    <rdfs:subClassOf rdf:resource="&RobotsAutomation;SUMO:Object"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#SUMO:Relation -->

<owl:Class rdf:about="&RobotsAutomation;SUMO:Relation">
    <rdfs:subClassOf rdf:resource="&RobotsAutomation;SUMO:Abstract"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#SUMO:TimeInterval -->

<owl:Class rdf:about="&RobotsAutomation;SUMO:TimeInterval">
    <rdfs:subClassOf rdf:resource="&RobotsAutomation;SUMO:TimeMeasure"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#SUMO:TimeMeasure -->

<owl:Class rdf:about="&RobotsAutomation;SUMO:TimeMeasure">
    <rdfs:subClassOf rdf:resource="&RobotsAutomation;SUMO:PhysicalQuantity"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#SUMO:TimePoint -->

<owl:Class rdf:about="&RobotsAutomation;SUMO:TimePoint">
    <rdfs:subClassOf rdf:resource="&RobotsAutomation;SUMO:TimeMeasure"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#Semi-AutonomousRobot -->

<owl:Class rdf:about="&RobotsAutomation;Semi-AutonomousRobot">
    <rdfs:subClassOf rdf:resource="&RobotsAutomation;Robot"/>
</owl:Class>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#SimpleRoboticSystem -->

<owl:Class rdf:about="&RobotsAutomation;SimpleRoboticSystem">
    <rdfs:subClassOf rdf:resource="&RobotsAutomation;RoboticSystem"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="&RobotsAutomation;SUMO:properPart"/>
            <owl:onClass rdf:resource="&RobotsAutomation;Robot"/>
            <owl:qualifiedCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

```
    <!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#Transformation -->

    <owl:Class rdf:about="&RobotsAutomation;Transformation">
        <rdfs:subClassOf rdf:resource="&RobotsAutomation;SUMO:Function"/>
    </owl:Class>
</rdf:RDF>



<!-- Generated by the OWL API (version 3.4.2) http://owlapi.sourceforge.net -->
```

And some individuals for testing:

```
<!--
    ///////////////////////////////////////////////////////////////////////////////////////
    //
    // Individuals
    //
    ///////////////////////////////////////////////////////////////////////////////////////
     -->



    <!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#R1 -->

    <owl:NamedIndividual rdf:about="&RobotsAutomation;R1">
        <rdf:type rdf:resource="&RobotsAutomation;AutonomousRobot"/>
        <pos rdf:resource="&RobotsAutomation;R1CartesianPositionPoint"/>
        <pos rdf:resource="&RobotsAutomation;R1SeenByR2CartesianPositionPoint"/>
        <workIn rdf:resource="&RobotsAutomation;explorationRobotGroup"/>
    </owl:NamedIndividual>



    <!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#R1CartesianPositionPoint -->

    <owl:NamedIndividual rdf:about="&RobotsAutomation;R1CartesianPositionPoint">
        <rdf:type rdf:resource="&RobotsAutomation;CartesianPositionPoint"/>
        <cartesianX rdf:datatype="&owl;real">0</cartesianX>
        <cartesianY rdf:datatype="&owl;real">0</cartesianY>
        <in rdf:resource="&RobotsAutomation;R1CoordinateSystem"/>
    </owl:NamedIndividual>



    <!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#R1CoordinateSystem -->

    <owl:NamedIndividual rdf:about="&RobotsAutomation;R1CoordinateSystem">
        <rdf:type rdf:resource="&RobotsAutomation;CartesianCoordinateSystem"/>
        <ref rdf:resource="&RobotsAutomation;R1"/>
    </owl:NamedIndividual>



    <!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#R1SeenByR2CartesianPositionPoint -->

    <owl:NamedIndividual rdf:about="&RobotsAutomation;R1SeenByR2CartesianPositionPoint">
        <rdf:type rdf:resource="&RobotsAutomation;CartesianPositionPoint"/>
        <cartesianX rdf:datatype="&owl;real">-1000</cartesianX>
        <cartesianY rdf:datatype="&owl;real">0</cartesianY>
        <in rdf:resource="&RobotsAutomation;R2CoordinateSystem"/>
    </owl:NamedIndividual>



    <!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#R2 -->

    <owl:NamedIndividual rdf:about="&RobotsAutomation;R2">
        <rdf:type rdf:resource="&RobotsAutomation;AutonomousRobot"/>
        <pos rdf:resource="&RobotsAutomation;R2CartesianPositionPoint"/>
        <pos rdf:resource="&RobotsAutomation;R2SeenByR1CartesianPositionPoint"/>
        <workIn rdf:resource="&RobotsAutomation;explorationRobotGroup"/>
    </owl:NamedIndividual>



    <!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#R2CartesianPositionPoint -->

    <owl:NamedIndividual rdf:about="&RobotsAutomation;R2CartesianPositionPoint">
        <rdf:type rdf:resource="&RobotsAutomation;CartesianPositionPoint"/>
        <cartesianY rdf:datatype="&owl;real">0</cartesianY>
        <cartesianX rdf:datatype="&owl;real">0</cartesianX>
        <in rdf:resource="&RobotsAutomation;R2CoordinateSystem"/>
    </owl:NamedIndividual>



    <!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#R2CoordinateSystem -->

    <owl:NamedIndividual rdf:about="&RobotsAutomation;R2CoordinateSystem">
        <rdf:type rdf:resource="&RobotsAutomation;CartesianCoordinateSystem"/>
        <ref rdf:resource="&RobotsAutomation;R2"/>
    </owl:NamedIndividual>
```

```
<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#R2SeenByR1CartesianPositionPoint -->

<owl:NamedIndividual rdf:about="&RobotsAutomation;R2SeenByR1CartesianPositionPoint">
    <rdf:type rdf:resource="&RobotsAutomation;CartesianPositionPoint"/>
    <cartesianY rdf:datatype="&owl;real">0</cartesianY>
    <cartesianX rdf:datatype="&owl;real">1000</cartesianX>
    <in rdf:resource="&RobotsAutomation;R1CoordinateSystem"/>
</owl:NamedIndividual>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#cartesianBackOfOperator -->

<owl:NamedIndividual rdf:about="&RobotsAutomation;cartesianBackOfOperator"/>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#cartesianFrontOfOperator -->

<owl:NamedIndividual rdf:about="&RobotsAutomation;cartesianFrontOfOperator"/>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#cartesianLeftOfOperator -->

<owl:NamedIndividual rdf:about="&RobotsAutomation;cartesianLeftOfOperator"/>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#cartesianRightOfOperator -->

<owl:NamedIndividual rdf:about="&RobotsAutomation;cartesianRightOfOperator"/>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#explorationEnvironment -->

<owl:NamedIndividual rdf:about="&RobotsAutomation;explorationEnvironment">
    <rdf:type rdf:resource="&RobotsAutomation;RoboticEnvironment"/>
    <equippedWith rdf:resource="&RobotsAutomation;explorationRoboticSystem"/>
</owl:NamedIndividual>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#explorationRobotGroup -->

<owl:NamedIndividual rdf:about="&RobotsAutomation;explorationRobotGroup">
    <rdf:type rdf:resource="&RobotsAutomation;RobotGroup"/>
    <SUMO:properPart rdf:resource="&RobotsAutomation;explorationRoboticSystem"/>
</owl:NamedIndividual>



<!-- http://www.semanticweb.org/ontologies/2013/7/RobotsAutomation.owl#explorationRoboticSystem -->

<owl:NamedIndividual rdf:about="&RobotsAutomation;explorationRoboticSystem">
    <rdf:type rdf:resource="&RobotsAutomation;CollectiveRoboticSystem"/>
</owl:NamedIndividual>
```