Universität Bamberg

Thema: **Semantic Modeling and consistency checking of location-based games**

Semantische Modellierung und Konsistenzprüfung ortsbezogener Spiele

Bachelorarbeit

im Studiengang Angewandte Informatik der Fakultät Wirtschaftsinformatik und Angewandte Informatik der Otto-Friedrich-Universität Bamberg

Verfasser: Clemens Klug
Gutachter: Prof. Dr. Christoph Schlieder

Abgabedatum: February 24, 2016

Contents

1. Introduction 1
1.1. Problem definition 2
1.2. Scope of this work 2

2. Related work 4
2.1. Location-based games 4
2.2. Game design patterns 5
2.3. Spatial representation and reasoning in semantic ontologies 8
2.3.1. W3C Basic Geo Vocabulary 8
2.3.2. LinkedGeoData.org 8
2.3.3. SpaceOntology 9
2.3.4. Region Connection Calculus 9
2.3.5. GeoSPARQL 10
2.3.6. PelletSpatial 11

3. Solution approach 12
3.1. Methodology 12
3.2. The ontology 12
3.2.1. Encoding constraints 12
3.2.2. Spatial representation 13
3.2.3. Gameplay ontology 15
3.3. Select a reasoner 17
3.4. What can not be modeled 17

4. Implementation 19
4.1. Model of the ontology 19
4.2. Augment the reasoner 24
4.3. Extended testing 26
4.4. Implementing a game 27

5. Evaluation 28
5.1. GeoTTT 28
5.2. Neocartographer 28
5.3. CityPoker 29
5.4. Metrics of different game implementations 29
5.5. Snapshots 30

6. Discussion and future work **31**

References **33**

A. Appendix **35**
A.1. Ontology models 35

List of Figures **38**

List of Tables **39**

List of Algorithms **40**

List of Abbreviations **41**

List of Symbols **42**

Eidesstattliche Erklärung **43**

1 Introduction

With the current ubiquity of handheld or wearable computational devices, mobile games partake in many peoples everydays life. As nearly every such mobile device is equipped with positioning systems, mobile games considering the spatial position of the player are a widespread occupation, with geocaching being a very popular and simple type. Such location based games appear in a broad variety, from finding more or less well hidden caches in geocaching to more sophisticated games with embedded educational aspects. For example, the game Neocartographer is aimed at improving the players' understanding of Voronoi tessellation and spatial thinking in general [1]. With the addition of a spatial context, a game is played in an area called gamefield where the game actions happen. This gamefield needs to be configurable to be playable in more than one place. With many different games come very different requirements for such a configuration format. While a fairly simple game configuration of Geocaching can easily be stored in any markup language being able to store coordinates along with some commentary like GPX[1], any elaborated game needs more information and structure. This information is typically stored in a game-specific configuration format. And while this reduces the memory footprint and enables a streamlined editing process, it prevents easy sharing and re-use of existing objects between multiple game types. Additionally, not only are the data formats game-specific, but all games also need to implement their own, custom validity and consistency checking. This problem can be solved by using a common, extensible format for game configurations. Such a format would also enable re-use of existing

[1] GPS Exchange Format

content creation and validation tools across different games. Additionally, when a formal representation is used, high-level specification of validation is made possible.

The goal of this thesis is the development of such a configuration syntax using semantic modeling and evaluating the possibilities of validation with the established building blocks from the semantic web stack.

1.1. Problem definition

Ever since the creation, mobile, location-based games suffer from this configuration overhead. Even mobile games built using the current framework of the Geogames team at the University of Bamberg[2] all implement their own solutions to load their respective game configurations from XML or JSON based files. Realizing a common solution to read, parse, and verify configurations will reduce the effort to rebuild this part for each future game. A generalized framework developed by ARIS[3] allows the creation of certain, fairly simple game types. To prevent introducing such unwanted limitations on the design for future games, such an approach needs to respect the design patterns for mobile games, further explained in section 2.2.

With ontologies as open, extensible formats, such a vocabulary can easily be extended to support unforeseen game designs. An ontology built using the semantic web stack with the Resource Description Framework (RDF) and Web Ontology Language (OWL) enables linking to and reuse of resources in Linked Open Data projects, e.g. referring to spatial objects from the LinkedGeoData[4] project. Not only can this connection be used to create games embedded in Linked Open Data, closing the feedback loop introduces curation of the linked datasets by the players of the location-based games [2, 3].

1.2. Scope of this work

The scope of this work is to provide a representation of snapshots from location-based games based on semantic modelling in OWL 2 Direct Semantics (successor to OWL Description Logic OWL-DL). This will be coupled with a validation service to ensure the

[2] http://www.geogames-team.org/
[3] http://arisgames.org/
[4] http://linkedgeodata.org

consistency of these models. Representing and validating temporal relations and constraints will not be considered here. There is also no attempt to model the actual game mechanics, i.e. as instructions, into the ontology. In summary, representation and validation will be limited to static snapshots of game state, with dynamic features such as state change left open.

The state model will be based on the game state representation of various existing geogames (GeoTicTacToe, CityPoker, Neocartographer). Game state and constraints will be modelled using OWL restrictions. The limits of validating these restrictions using a reasoner will be shown as well as an augmented solution to fix these cases and introduce a spatial dimension. Finally the implementation of some games will serve as evaluation to prove the flexibility of this approach.

2

Related work

2.1. Location-based games

As defined in [4], a location-based game (Geogame) can be described by a set G defined as:

$$G = (S, A, time, value, sync)$$

S Set of game states. A state itself is built from mappings between players (P), locations (L) and resources (R), such that $s = (P \to L, R \to L \cup P)$.

A Spatio-temporal actions embedding information about a games mechanics, as they define transitions between any states by creating a mapping between two states: $a : S \to S$.

$time$ Every action consumes time described by mapping actions against time with $time : A \to \mathbb{R}^{+}$.

$value$ Level of interest of players for states: $value : S \to V$. This mapping encodes a value for states, e.g. when a certain state means a player has won the game.

$sync$ Denotes the speed factor of the game in form of a synchronization interval

Besides these definitions, two basic invariables of spatial and temporal coherence are postulated. While the aspects of temporal coherence are covered by the entry of $time$ in the geogame definition G, spatial coherence is a run time invariant, restricting the ability for interaction between players and resources exclusively to the player's current location.

2.2. Game design patterns

Much like ontologies, design patterns enable communication about a subject with a common set of concepts and define practical approaches for common problems. Originating from architectural science, these benefits soon were adapted to the field of software engineering. As modern-days games are virtually all software based — even classic cardboard games get enhanced with computational devices [5] — game designers and developers get to learn the software patterns as a useful utility not only to avoid misunderstandings [6]. Given the complexity of designing games, a set of patterns for game design was developed, followed by an extension for mobile games [7, 8]. These game design patterns for mobile games from 2004 do not aim only at location-based mobile games, rather any game playable on a mobile platform. The dedicated game design pattern language for location-based games shown in Figure 2.1 are compiled in [9].

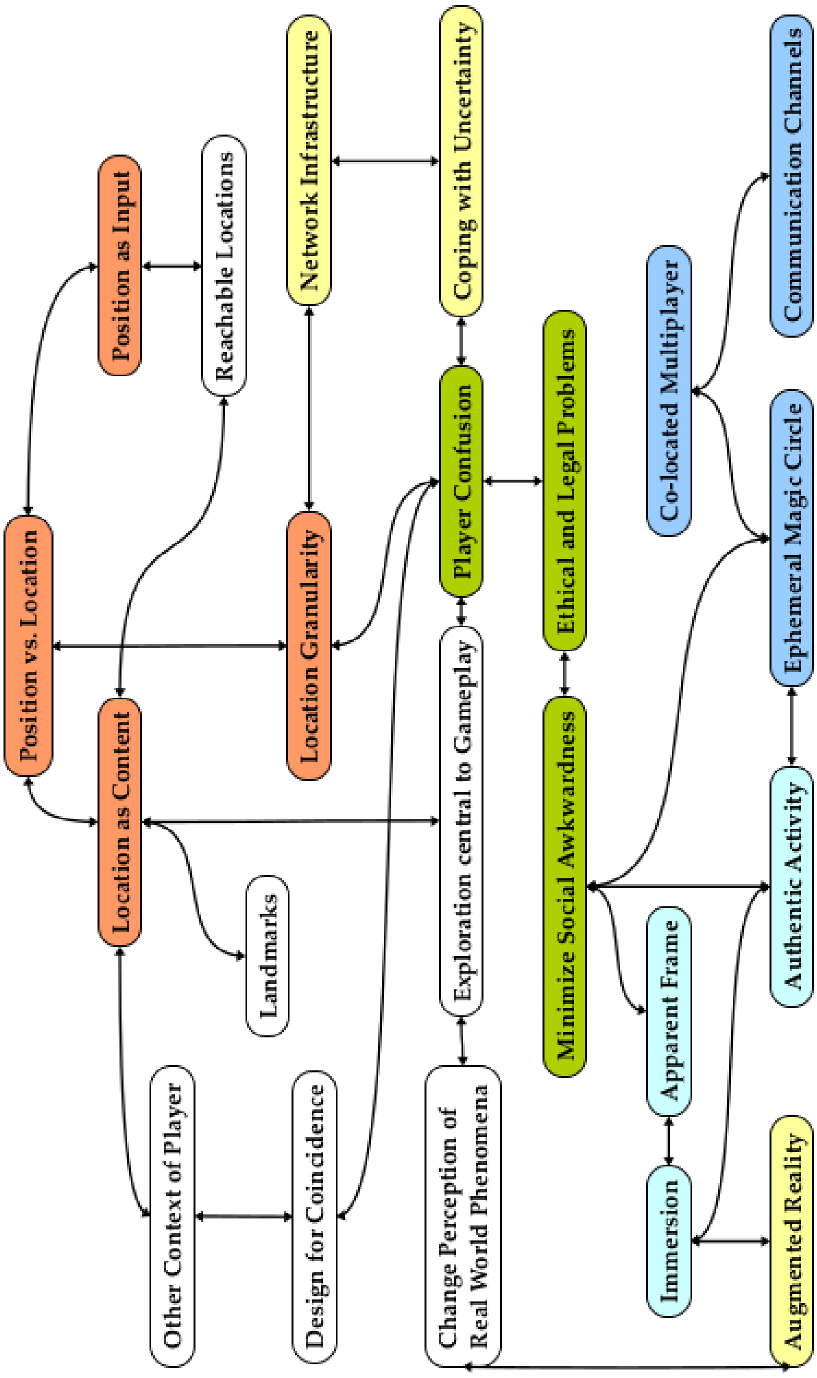

Figure 2.1.: Overview of games design patterns relevant for location-based games [9]

The graphical overview (Figure 2.1) not only shows related patterns, but groups patterns by common properties: spatial patterns (orange), playing in public spaces (green), immersive gameplay (light blue), technology (yellow), multiplayer (dark blue) and physical world related issues (white).

While these patterns do provide a descriptive framework, the meaning and intentions behind them can differ, especially between designers and developers of games. The latter ones have to deal with rather small chunks of game mechanics, compared to design patterns. Translations of one pattern into a set of mechanics depends heavily on the understood intentionality behind the patterns and mechanics. Adding a contextualization layer between the highly abstract patterns and the more concrete mechanics as shown in Figure 2.2 both situates general design patterns in new designs and clarifies the intentionality behind outlined game mechanics.

Figure 2.2.: Abstraction levels of game design [6]

With such a contextualization, scenarios showcase intended usages. These scenarios in turn act as indicator to analyze the compliance of game mechanics for the design goals. Figure 2.3 displays the role of contextualization in the design and development process of a game. The abstract patterns are instantiated with some meaning by the contextualization, which gets broken down to interaction handling mechanics. Finally, the mechanics are implemented in code [6].

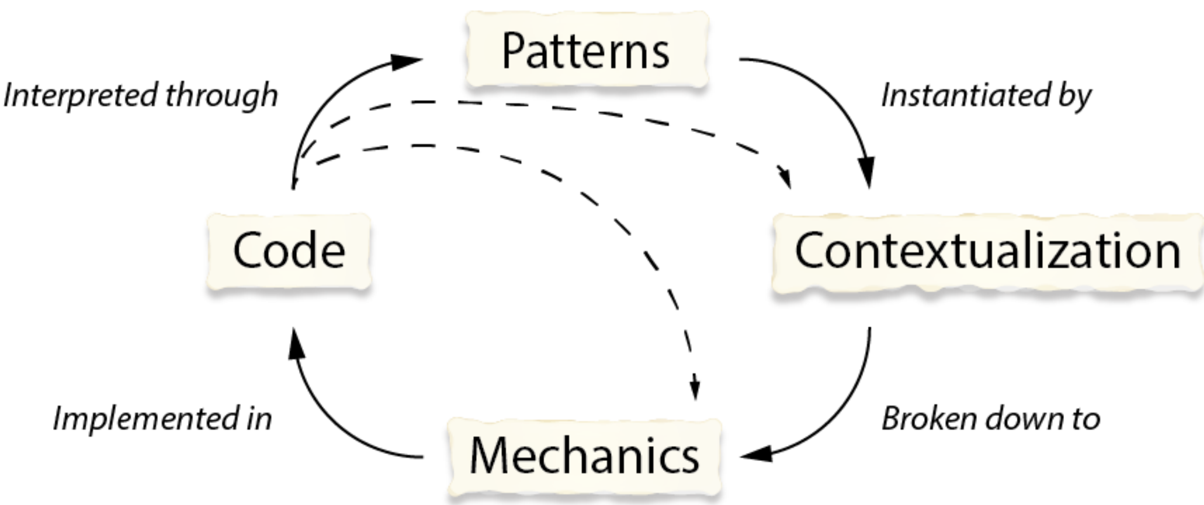

Figure 2.3.: Conceptual relations [6]

2.3. Spatial representation and reasoning in semantic ontologies

When representation of and reasoning about spatial data sets in semantic ontologies is needed, there are multiple problems to solve. First, one has to choose whether topological information and spatial reasoning with qualitative spatial relations, spatial geometry and reasoning with spatial operators, or both is required. Afterwards, a representation able to encode the needed information has to be found and integrated. The last part of the puzzle is choosing a reasoner and triple store having adequate capabilities like GeoSPARQL [10].

2.3.1. W3C Basic Geo Vocabulary

The Semantic Web Interest Group of the World Wide Web Consortium (W3C) has worked on a basic Geo Vocabulary[1] for representation of spatial objects regarding their position to the latest World Geodetic System standard (WGS84). This vocabulary does not define any relations between such spatial objects, however it does provide a relation to *locate* any individual to a point. Although it has not been subjected to the standardization process of the W3C, and therefore has not been reviewed and discussed on, it is already widely in use [11, 12].

Both GeoOWL and the NeoGeo Geometry Ontology (Geovocab) extend this vocabulary. GeoOWL is a basic ontology for annotating web resources with geospatial properties. It includes a classification of features into four basic geometries (point, line, box, and polygon) along with some general properties like elevation and radius [13]. The Geovocab further elaborates this geometrical expressiveness with more complex aggregates [14].

2.3.2. LinkedGeoData.org

LinkedGeoData.org is a Linked Open Data project, which extracts data from the OpenStreetMap[2] project and represents it in the RDF format. Spatial representations are necessary to represent the coordinates of the described features. This project uses the WGS84 model proposition by the W3C already mentioned. Additionally, a *close-by* rela-

[1] https://www.w3.org/2003/01/geo/
[2] https://www.openstreetmap.org

tion is used without any further clarification [15, 16].

2.3.3. SpaceOntology

Striving for representing spatial information of different levels of abstraction for purposes of planning, SpaceOntology was developed. An illustration is shown in Figure 2.4 with a sample instance of a relation. The use case of spatial planning determined the representation of topological relations in respect to both explicit horizontal and vertical axes as well as either fuzzy or numeric distances [17].

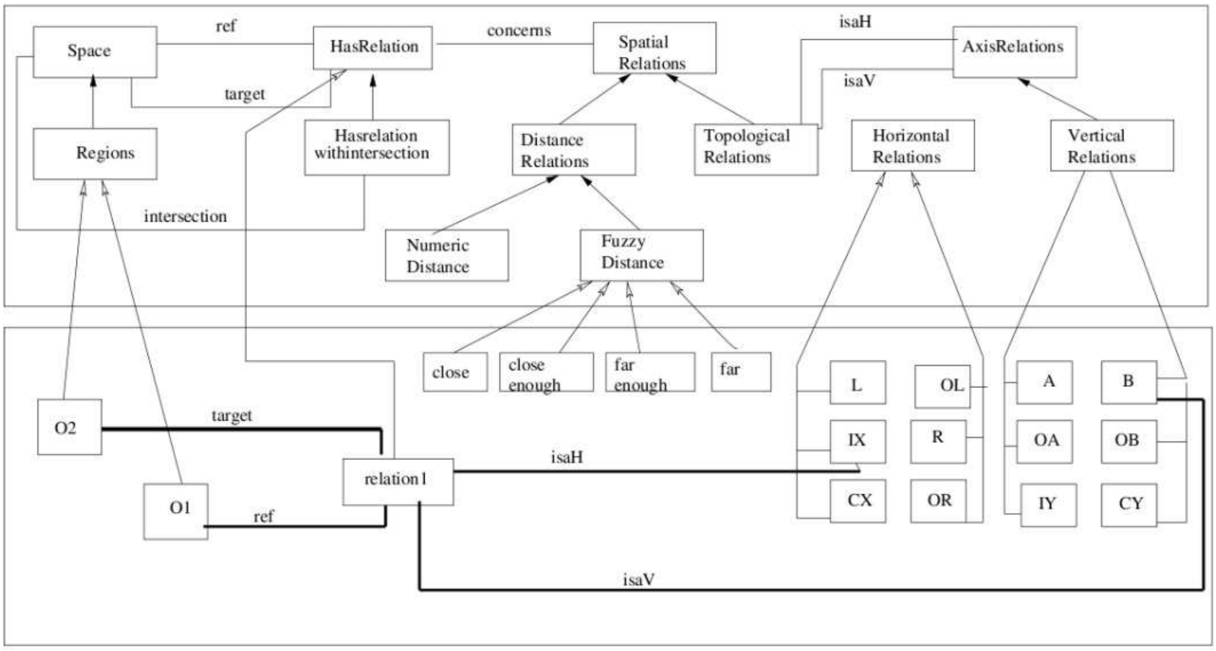

Figure 2.4.: SpaceOntology [17]

2.3.4. Region Connection Calculus

The Region Connection Calculus (RCC8) defines eight basic topological relations between two regions depicted in Figure 2.5. By expressing disconnected, externally connected, partially overlapping, equal as well as tangential and non-tangential proper parts and the inverses of the latter two, any topological relation can be expressed, the calculus is jointly exhaustive and pairwise disjoint (JEPD). A relaxation regarding the special cases of differentiation whether the edges or the content of two regions meet/overlap with just

5 relation is called RCC5; formerly externally connected regions are disconnected and there is only proper part respectively its inverse left [18]. As this calculus does not fix the concept of the regions to spatial objects, it has successfully been used to describe the relations of conceptual spaces [19].

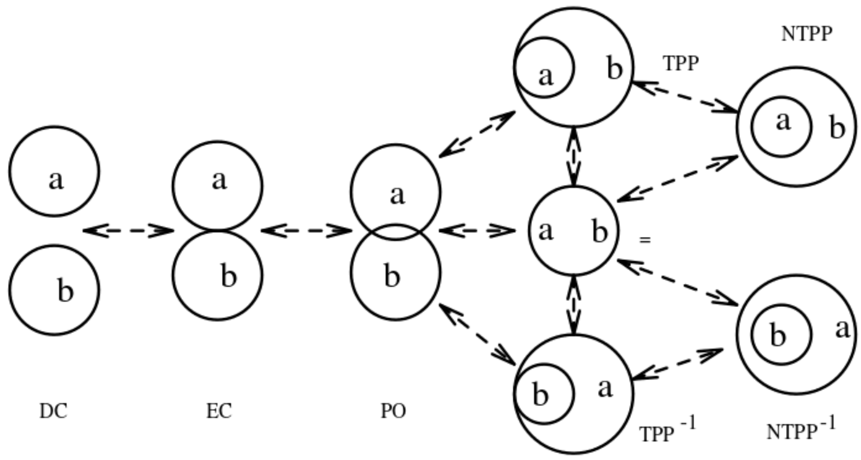

Figure 2.5.: RCC8 [18]

A representation of such qualitative spatial information in OWL-DL is shown in [20]. A translation of RCC relations to OWL is proposed along with an extension of OWL-DL with reflexive relations later added in OWL2-DL. This translation is built upon the creation of additional concepts expressing relationships between boundaries and contents of regions for each spatial relation between two regions, leading to a possibly huge number of additional classes and individuals.

2.3.5. GeoSPARQL

GeoSPARQL is an extension of the SPARQL language defined by the Open Geospatial Consortium (OGC) for supporting geospatial data on the Semantic Web. It is built up by several modules, each addressing a specific requirements class. The OWL classes for spatial objects are defined in a *core* component, while properties regarding topological relations between objects are outlined by a *topology vocabulary*. Other modules define geometries, topological query functions, implicit RDF triples and query rewrites [21, 22].

2.3.6. PelletSpatial

PelletSpatial is a extension for the Pellet[3] reasoner. It enables to formulate plain SPARQL queries about spatial relations as well as non-spatial semantic relations. Because of performance issues, it was necessary to extend the reasoner with a path-consistency algorithm dedicated to spatial relations stored separated from usual semantic relations handled by the base reasoner [23].

[3] https://github.com/Complexible/pellet

3

Solution approach

3.1. Methodology

Using the relevant information collected above, it is possible to structure an approach for building an OWL ontology suitable for solving the requirements from section 1.1. For validation purposes of a modeled geogame, a framework for encoding constraints has to be developed. Secondly, a useful spatial representation will enable both representation of locations as well as validation of their consistency with game-specific rules. Finally, a suitable OWL reasoner has to be selected and the limitations of using OWL reasoning for constraint-checking have to be evaluated.

3.2. The ontology

3.2.1. Encoding constraints

The definition of OWL 2[1] has four entries for expressing restrictions on classes:

- Object Property Restrictions

  This restriction is used to state existential and universal quantification on properties relating two individuals. Additionally, a restriction for a concrete individual as only object for a property can be created.

- Object Property Cardinality Restrictions

[1] https://www.w3.org/TR/owl2-syntax

When all individuals of a class shall have an amount of properties with individuals as object whether at least, exactly, or at most, object property cardinality restrictions can be used.

- Data Property Restrictions

  As with object properties, this restriction is used when properties with literals are considered.

- Data Property Cardinality Restrictions

  Analogously, this restriction states a numerical amount for properties with literals.

These restrictions are applied by a reasoner which will sort individuals into according classes, and will report any inconsistencies in the knowledge base [24]. Using restrictions for modeling constraints allows defining a sane base model while allowing further extensions and finer specifications through creating subclasses with more concrete restrictions for a limited subset of games. The use of subclassing also enables setting different constraints for individuals of the same base type within the same model.

Because the ontology developed in this work does not consider runtime updates and temporal aspects, some important aspects of gameplay, e.g. the number of players admissible, will further be encoded as data properties.

Given the open world semantics, restrictions committed to ensure a minimal amount of relations usually do not lead to an inconsistent knowledge base when not satisfied. The proposed solution of limiting the universe to the known individuals[2] did during the research for this work not lead to reproducible success, so an augmentation of the reasoner with SPARQL queries and logic implemented in Java were applied (see section 4.2).

3.2.2. Spatial representation

When choosing a representation for spatial features, the following properties were considered:

- Abstraction

  In general, the level of abstraction for both objects and constraints in the general framework should be as high as possible. This allows the game designer to formulate

[2] https://github.com/Complexible/pellet/wiki/FAQ#does-pellet-support-closed-world-reasoning

abstract constraints that need to hold for all games of a specific type. At the same time, the creation of concrete game fields is not overly restricted.

- Topology

  Abstract relations between game board elements and players between themselves and each others need the possibility to declare topological relations.

- Coordinates

  When a concrete instance of a geogame is to be configured, encoding of concrete coordinates is inevitable for a playable game. This is important as e.g. RCC-5 only deals with relations between spatial objects not with their concrete spatial extent.

- Availability

  Finally, the best ontology is useless when there is no way to actually put in in use because it is not accessible to the general public.

Table 3.1 shows these properties of the spatial representations outlined in section 2.3.

Ontology	abstraction	topology	coordinates	availability
W3C-WGS84	low	X	✓	✓
LinkedGeoData	low	(✓)	✓	✓(W3C-WGS84)
SpaceOntology	mid	✓	X	X
RCC8	high	✓	X	✓
GeoSPARQL	high	✓	✓	(✓)

Table 3.1.: Features of spatial representations

With a rather low abstraction level, W3C-WGS84 and the model used by LinkedGeoData are straightforwardly usable for representation of coordinates, however lack at supporting topological relations with no (W3C-WGS84), respectively one (LinkedGeoData) relation. The availability and wide usage of W3C-WGS84 promises great support by toolkits.

SpaceOntology has a more elaborate approach on encoding topologies, but suffers from a lack of representability for coordinates, so W3C-WGS84 has to be used to fill this gap. Anyhow, the missing availability at the time of this writing does rule this ontology out.

Like SpaceOntology, RCC8 focuses on topological relations, however it does extend to a higher level of abstraction, as there are no relations regarding any axis.

Made for querying spatial datasets, GeoSPARQL obviously supports representation of topologies and coordinates. Being primarily a query language, it relies on the basic building blocks, W3C-WGS84 and RCC8 relations. However, the standard offers multiple feature subsets leading to problems when using GeoSPARQL with different knowledge bases implementing different subsets [21, 25]. Therefore, no GeoSPARQL queries will be used, however the topological relations defined will be used together with coordinates provided by the W3C-WGS84 basic vocabulary.

Considering the circumstances of this work, complex spatial features will not be implemented for a maintainable level of complexity.

Although subsection 2.3.4 shows an approach for reasoning over RCC8 relations, [26] describes how implementing it leads to a huge amount of additional TBox statements, making it hard to integrate in other ontologies and dramatically limiting the complexity of solvable problems. An alternative approach utilizing PelletSpatial is described, however, due to the unknown whereabouts of PelletSpatial this does not work anymore either. What does exist and work are ontologies describing sub-properties and disjointedness between RCC8 relations.[3] However, such an approach does not add any valuable information in this case.

3.2.3. Gameplay ontology

Requirements

According to [4], an ontology for snapshots of geogames needs to consider five fields: actions as means of interaction, game conditions to determine states of the game, and locations, resources and players. These components need to be able to represent the relevant game design patterns for location-based games (see Figure 2.1). First, patterns relevant for modeling have to be distinguished from patterns relevant for design of concrete game instances or implementation of game engines, with a result listed in Table 3.2.

Category	Pattern	Required in model
Location	Position vs. Location	✓

[3] http://www.informatik.uni-bremen.de/~joana/ontology/modSpace/RCC-Ontology.owl

	Location as Content	✓
	Position as Input	✓
	Location Granularity	X
Technology	Network Infrastructure	X
	Coping with Uncertainty	X
	Augmented Reality	X
Public Space	Player Confusion	X
	Ethical and Legal Problems	X
	Minimize Social Awkwardness	X
Immersion	Immersion	X
	Apparent Frame	X
	Authentic Activity	X
Multiplayer	Ephemeral Magic Circle	X
	Co-located Multiplayer	(✓)
	Communication Channels	X
Physical world	Reachable Locations	X
	Landmarks	✓
	Other Context of Player	X
	Design for Coincidence	X
	Exploration central to Gameplay	(✓)
	Change Perception of Real World Phenomena	X

Table 3.2.: Evaluation of game design patterns regarding relevance for a geogame model

While most of these patterns are clearly in the scope of a geogame designer and others are particular for developers of game applications, a well-designed ontology can reduce expenditure. The integration of Landmarks for example can be encouraged by enabling relations to entries in OpenStreetMap, LinkedGeoData respectively. Even if additional relations are needed, these can easily be added through descriptions. While some patterns like Position as Input or Location as Content obviously need to be considered for an ontology, others like Exploration central to game might be surprising. This for example has to be considered, as it encourages giving as much freedom for players as possible to follow their own paths in an arbitrary order through the game field.

Typical progress of a game

The progress of most location-based games can be divided into three parts: pregame, ingame, postgame.

1. Pregame: The participants join the game. It is possible that one ingame player consists of a group of persons. After a briefing phase by the host, the players move to dedicated locations to start the ingame phase, and are equipped with necessary resources.

2. Ingame: The players perform the game according to the rules, interacting with caches and resources until a termination criterion is met.

3. Endgame: The game is over, and either a winner is declared or a draw was accomplished.

3.3. Select a reasoner

PelletSpatial (see subsection 2.3.6) seems like the perfect reasoner for this task, given the implementation of RCC relations. However, this tool is no longer available and therefore is not considered anymore.

As this work is based solely on free and open source software, a reasoner fitting into this environment is needed. Having support for the build management tool Maven[4] the Pellet reasoner[5] does not only fit the license requirements, it also ensures a streamlined build process without the need for manual installation and configuration of dependencies and obtaining software licenses. The Jena RDF API[6] needed for interfacing pellet from Java is present in the central maven repositories[7] with all these benefits, too.

3.4. What can not be modeled

As proposed, the ontology can only be used to describe snapshots of a geogame, such that no temporal aspects can be included. Besides that, it is not possible to express

[4] https://maven.apache.org/
[5] https://github.com/Complexible/pellet
[6] https://jena.apache.org/
[7] https://jena.apache.org/download/maven.html

conditional constraints like “If a Cache was occupied by a player, no more interactions are possible.” While such constraints can not be described directly, it is possible to encode them using a descriptive subclass interpreted differently by an implementation of a game — just like a strategy pattern in software engineering. However, this circumvents the idea of an automated validation as an implementation might just as well ditch this hint. Additionally, calculating numeric evaluations of game states in the ontology is not feasible and has to rely on an implementation by modeling a process instruction.

4
Implementation

4.1. Model of the ontology

The free and open-source ontology editor Protégé[1] was used for modeling the abstract requirements formulated earlier (see chapter 3) in OWL. Following these requirements, the hierarchy depicted in Figure 4.1 emerges.

Thereby a geogame is modeled by five submodules and one base element grouping the submodules to a specific game. The submodules represent players, resources, actions defining interactions between players and resources, spatial information and gameplay information in form of game conditions describing game state transitions.

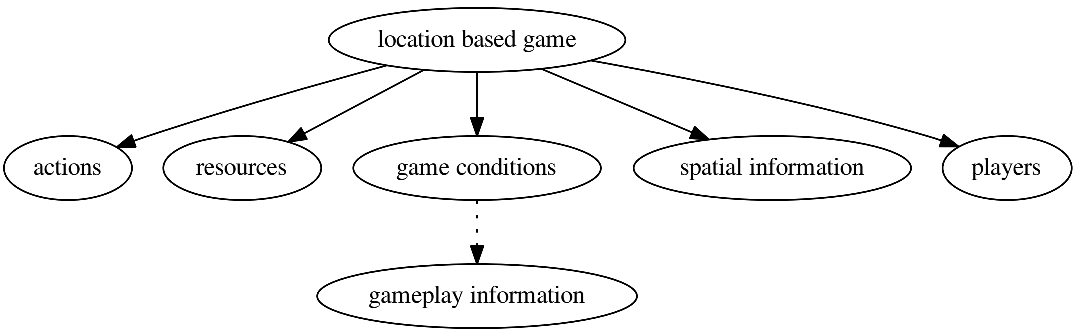

Figure 4.1.: Structure of the ontology

[1] http://protege.stanford.edu/

Actions and locations

As explained in section 2.1, interactions between players and resources shall only occur at the player's current location, therefore introducing spatial information as dependency of the definition of these interactions implicitly enforces such a behavior. An Action defines a conceptual cache, which can actually be located at multiple physical cache locations. Given the limitations of this work, locations can only be represented as Points according to the W3C WGS84 model, augmented with a GeoRSS radius and RCC8 relations from GeoSPARQL.

The amount of interactions possible at any given action of the analyzed games can be classified into one of three types: story telling, enabling and disabling other actions, and interactions of the player with transfer of resources. While story telling and enabling other actions only requires a dedicated subclass, interactions regarding resources need further definition deferred to a dedicated TokenHandler, as shown in Figure 4.2.

Questions as further challenges to solve before performing an action are deferred to an implementation and represented as plain strings, while synchronization times are essential for a balanced gameplay and therefore are encoded.

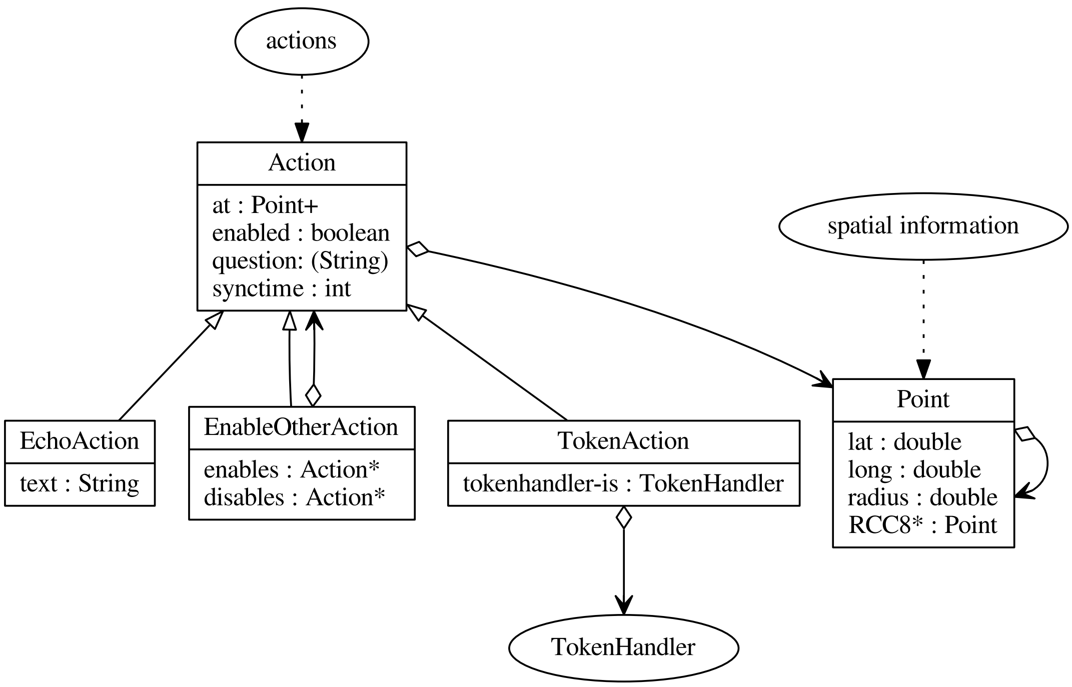

Figure 4.2.: Caches and actions

The base ontology includes restrictions for actions, such that all actions need to have a boolean flag to indicate whether an interaction is possible, a sync-time, and at least one Position to be situated in. Furthermore a restriction limits the type of referenced question to be a string.

The extended actions extend these restrictions by the following:

- EchoAction: A question has to be present, which will be echoed.
- EnabledOtherAction: Two properties enable or disable allow any amount of actions to be linked.
- TokenAction: A TokenHandler has to be linked.

Resources

The relations of resources in the ontology can be seen in Figure 4.3. Resources are represented by the class Token, with two subclasses: TokenSet, and Marker. While TokenSet is a valid Token containing several other tokens, Marker violates the requirements to be a proper geogame resource by being clonable only for tracking purposes (see TokenCount).

A TokenHandler specifies the behavior of an Action regarding the exchange of resources in the players inventory and the inventory of the cache. TokenCount and TokenCapture only accept tokens, while a TokenDispenser allows bidirectional transfer of tokens. The concept of TokenSets enables to adjust the amount of tokens dispensed at once by creating sets with the desired amount. TokenCount takes advantage of the violations of the Marker token for counting each appearance of a player by duplicating its identifying token and incrementing a counter. TokenCapture on the other hand relies on proper tokens and blocks the Actions it belongs to after one player has dropped a Token.

The TokenSet and all TokenHandlers are restricted to be able to hold any amount of tokens via the has-token property. The TokenDispenser has an additional property to ensure there is at least one token linked.

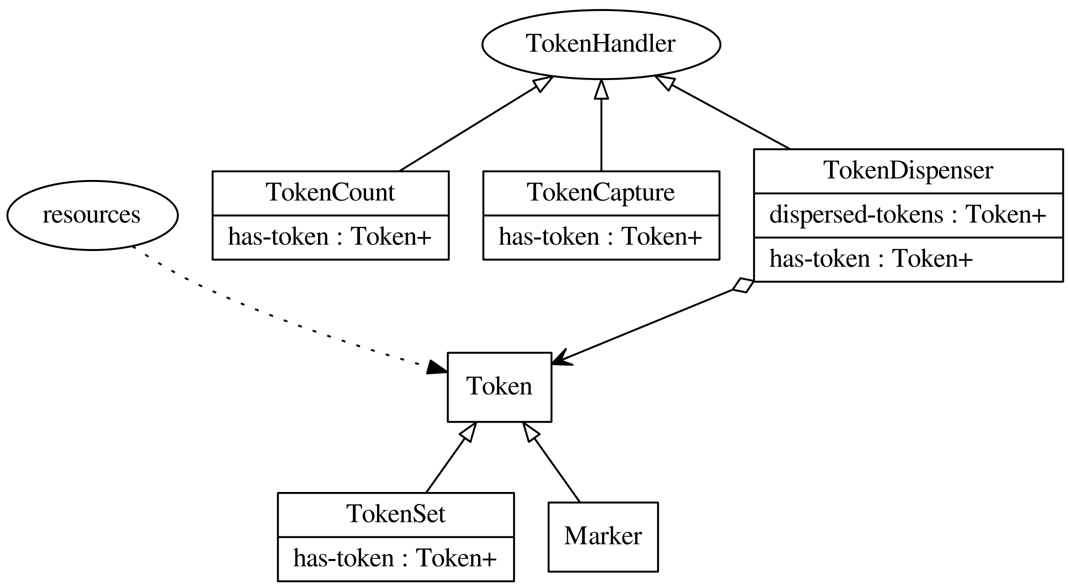

Figure 4.3.: Resources and their usage

Game conditions

Conditions are elements determining whether a state transition is permissible or not. Game state transitions between the states of not started, in game, and the two after-game states of a drawn and a won game are described by the three conditions DrawCondition, StartCondition and WinCondition visible in Figure 4.4. These conditions are based on logic conditions for supporting arbitrary combinations of conditions with an arbitrary choice of boolean combination modes.

Conditions not depending on other conditions are TimeOutCondition, TokenCondition and PlayerLocationCondition. The TimeOutCondition initially allows a transition while setting a timeout. Subsequent queries for permission will be denied until this timeout is over therefore allowing state transition afterwards. A PlayerLocationCondition will pass transition requests only if there are players at locations intersecting with the given locations. The TokenCondition refers to a list of TokenHandler and will permit transitions if a player has a Token dropped at every handler.

Any of the above conditions can have an assigned action which will be processed when the state transition occurs. Such actions are a handy tool to equip players with an initial set of resources or enable the ingame caches for interaction.

This is asserted by a restriction to the base condition, so that all conditions can link to an action. A LogicCondition has additional restrictions to limit objects linked as sub-conditions to conditions and a combination property with type string to encode the boolean combination mode. The PlayerLocationCondition needs to have at least one location linked, while a TimeOutCondition needs to have a non-negative integer value for indicating the duration of the timeout in seconds. Similarly, the TokenCondition is restricted to have a set of at least one TokenHandler.

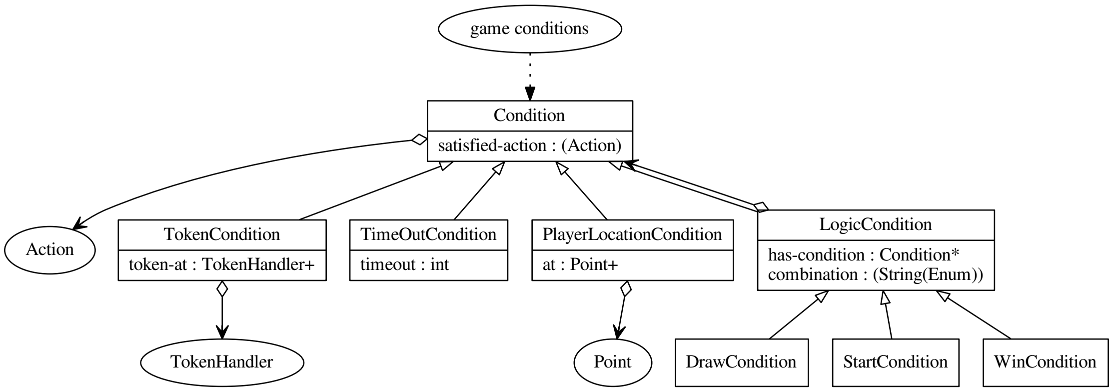

Figure 4.4.: Dynamic elements

Players

Modeling a player is rather straightforward with a position, a name and an inventory as a set of tokens (Figure 4.5).

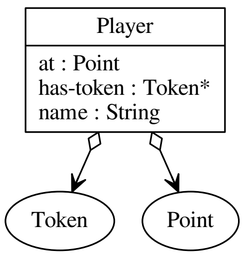

Figure 4.5.: Model for players

Complete game

The main individual representing a location-based game links all the afore-mentioned elements together as depicted in Figure 4.6. The inclusion of resources and locations is created implicitly through linking to actions with assigned token handlers with tokens and locations. Additionally, resources can analogously be contained in transition actions defined in draw, start and win conditions. Due to the snapshot nature of this representation, there may be a varying number of players present. Therefore, modeling the admissible amount of players through restrictions is not feasible. Instead, this is addressed by introducing a dedicated property indicating the admissible amount of players as well as references to possible present players.

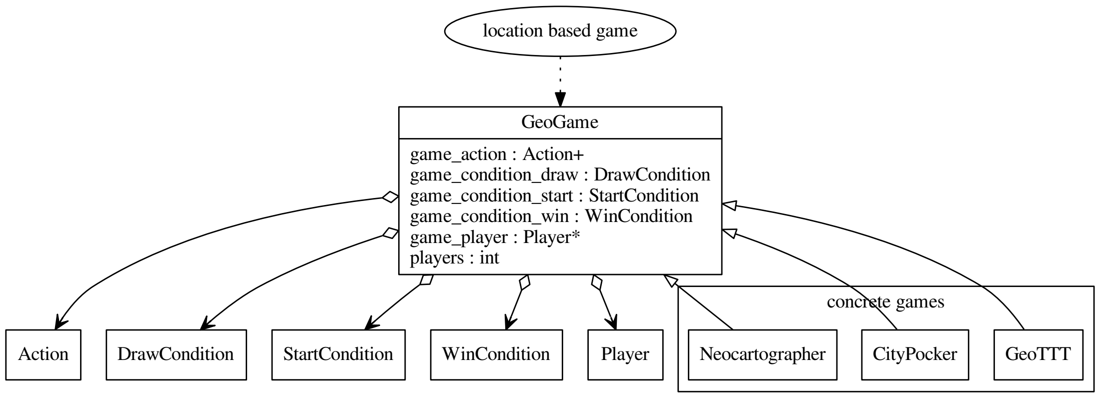

Figure 4.6.: Main individual representing a concrete game instance

This class only has a few restrictions to ensure type consistency of relations and limit the amount of each draw, start, and win condition to one, and requiring at least one game action, too.

4.2. Augment the reasoner

As during the implementation the approach of limiting the universe to known individuals for closed-world reasoning for asserting existential restrictions did reveal to be not reliable, a custom validation step outlined in Algorithm 4.1 was applied.

Algorithm 4.1 Validation algorithm

```
function VALIDATE(ontology)
    results ← emptyList()
    for all restriction ∈ ontology.restrictions do
        for all individual, property, object, count ∈ ontology.query(restriction) do
            if restriction.count ≠ count then
                results.add((individual, property, count))
            else if restriction.objectType ≠ object.type then
                results.add((individual, property, object.type))
            end if
        end for
    end for
    return results
end function
```

A spatial validation service validates RCC8 relations by calculating regions of given locations and verifying them against the stated spatial relations (see Algorithm 4.2).

Algorithm 4.2 Spatial validation algorithm

```
function VALIDATERCC(ontology, game)
    results ← emptyList()
    for all rcc ∈ RCC8 do
        for all point1, rcc, point2 ∈ ontology.query(game, rcc) do
            if rcc.validate(point1, point2) = ⊥ then
                results.add((point1, rcc, poin2))
            end if
        end for
    end for
    return results
end function
```

Additionally, it is not only able to calculate average distances over all locations to detect possibly misplaced locations, but also it determines the average distances of location groups, grouped by belonging to actions with a common token condition (see Algorithm 4.3).

Algorithm 4.3 Average group distance calculation

```
function GROUPDISTANCES(ontology, game)
    groups ← emptyList()
    conditions ← ontology.get(game).winCondition.subConditions
    for all condition ∈ conditions do
        if condition.type = TokenCondition then
            groups.add(conditions)
        else if condition.type ∈ LogicCondition then
            conditions.enque(condition.subConditions)
        end if
        conditions.deque(condition)
    end for
    conditions = ∅

    distances ← emptyList()
    for all group ∈ groups do
        results.add((group, averageDistance(group)))
    end for
    return results
end function
```

The implementation of all these validation steps in Java is displayed in Figure 4.7.

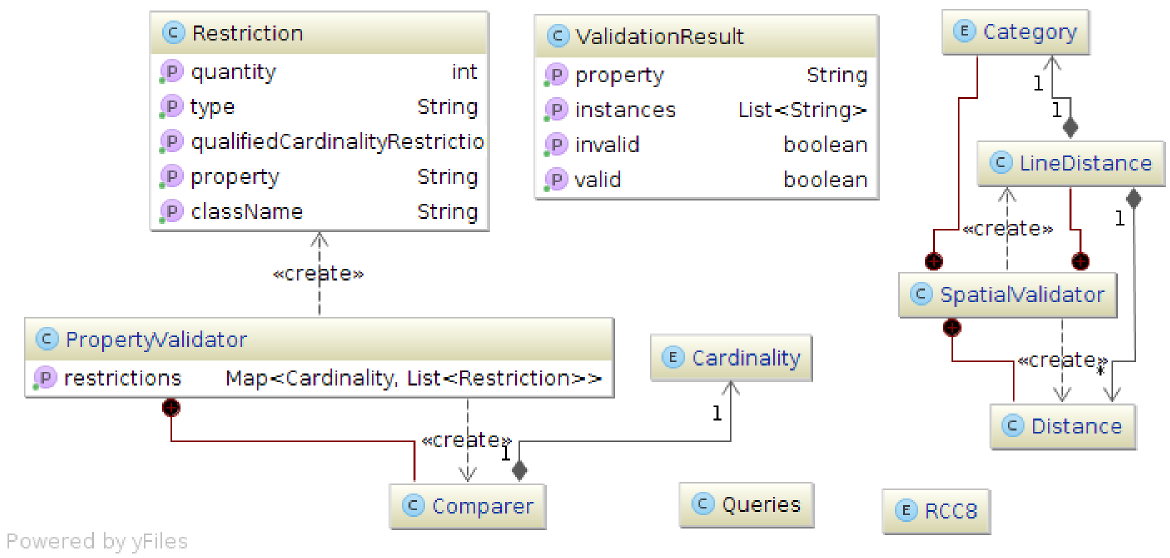

Figure 4.7.: Class diagram Validation

4.3. Extended testing

For empirical testing of the described ontology, an according Java mapping was created together with a loader interfacing Pellet through the Jena API. This loader can derive a geogame configuration object from a given ontology and a reference to the main geogame element in the ontology. By adding some logic to the mapped data classes and creating a game manager as well as a rather simple commandline interface (CLI), a playable game was achieved and tested. This in turn enabled the creation of automated non-interactive integration tests by using the JUnit[2] testing framework.

The limitations of this CLI were revealed when adding CityPoker and Neocartographer, as the CLI was developed with GeoTTT as an example, it was limited to this game. While for supporting CityPoker only an interaction mode for token dispensers was necessary, such an extension was delayed considering the facultative status of the CLI and testing features.

4.4. Implementing a game

Given this ontology, implementing respectively modeling a geogame is a rather straightforward process: A suitable class for actions is selected according to the desired interaction modes at caches and a subclass thereof is created for defining restrictions without affections on other games using the same base class. If there already is a suitable extended class with matching restrictions, it can be reused. The next step is the selection of tokens and token handlers for these actions following the same principle. By creating game specific subclasses it is possible to enforce usage of game specific elements. After modeling the rule set defining state transitions by building the start, draw, and win conditions analogously, the final step is the extension of the GeoGame class to build the whole game together.

The instantiating step follows after this modeling part and introduces an actually playable game configuration. By creating the needed individuals and properties, the previously proposed restrictions as well as the ones defined in the base ontology are satisfied and a valid knowledge base and game representation emerges.

This split into declaration and definition allows the onetime declaration of a location-based game with multiple instances defined upon.

[2]http://junit.org/

5 Evaluation

By implementing GeoTicTacToe (GeoTTT), CityPoker and Neocartographer the suitability of the previously defined ontology for modeling location-based games will be evaluated.

5.1. GeoTTT

As GeoTTT is played on a three-by-three grid of caches, the restriction of enforcing nine caches in the game is straightforward. Additionally, the participating players need to have the tokens needed for capturing a cache in their private inventories. By creating a start condition with a TokenDisperser action as satisfied action both the need for players to be present at a certain starting location as well as the initial distribution of resources is achieved. The time-out counter triggering a game draw is also started by this start condition. The game of GeoTTT relies on the mechanics of capturing caches forming a line of three caches owned by the same player. This winning condition is encoded as a set of Token-Conditions, each containing a line of three caches combined by a logical *AND* condition, i.e. exhaustive representation of all winning configurations.

5.2. Neocartographer

While the bridge mode of Neocartographer is straightforward implementable just like GeoTTT, the area mode on the other hand is not presentably being a numerical condition. It can implicitly be used by a game implementation by waiting until the time-out counter is over and measuring the areas afterwards. This can be achieved by combining the bridge-mode conditions and the time-out by a logical *OR* condition. Due to the variable amount of caches between nine and sixteen, the restrictions in the bridge-mode can vary greatly between two instances and cannot be fixed to a certain amount like in GeoTTT.

Besides that, the main difference to GeoTTT is a variable number of actions between nine and twelve. Additionally, a bounding box with a maximally admissible size and an upper limit for distances between neighboring caches is modeled.

5.3. CityPoker

Determining a winning player of CityPoker differs greatly from other location-based games because it is based on a valuation of the set of cards present in the player's inventory at the end of the game instead of spatial configurations. As with the area mode of Neocartographer, this can only be represented by having a time-out counter to indicate the end of the game and a game application with a token valuation and decision logic to pinpoint a winning player.

5.4. Metrics of different game implementations

A collection of metrics is listed in Table 5.1, namely the number of classes, individuals restrictions, and classes in use. The main ontology clearly has the highest amount of defined classes by establishing the class hierarchy to account for a large variety of games to be modeled. While there are obviously no individuals and therefore no classes in use, there is on average one restriction per class present. These restrictions are used to ensure the use of sane types on relations as well as a minimal amount of properties necessary for proper function of a class. Not listed is the amount of properties, namely 28, because there are no definitions of additional properties in the game-specific ontologies.

The model for GeoTTT (Code Listing A.1) adds 13 restrictions on nine different classes, while using one unmodified class from the base ontology. Therefrom, 59 individuals are created, nine each for cache locations, actions and token handlers, eight exhaustive winning configurations, six tokens for each player, with the rest accumulated by the main game element, conditions and the initial resource distribution mechanism.

CityPoker (Code Listing A.2) measures similar to GeoTTT with an additional restriction and three more unaltered classes in use.

Neocartographer (Code Listing A.3) however adds 24 additional restrictions on one extended class more, totaling to ten classes. The great increase of individuals and used classes indicates a higher complexity of the model originating from the two alternative game modes. Notably, the instance of Neocartographer used to collect the metrics is based on a minimal game field with nine out of 16 admissible caches and therefore indicates a lower limit of individuals (All 16 tokens per player are accounted for, though). There will be an increase of at least three individuals per additional cache plus a variable amount of bridge configurations.

Ontology	Classes	Individuals	Restrictions	Used classes
main	23	—	29	—
GeoTTT	9	59	13	10
CityPoker	9	62	14	13
Neocartographer	10	84	24	18

Table 5.1.: Ontology metrics

5.5. Snapshots

The creation of a snapshot is straightforward doable through instantiation of individuals of all classes necessary for a game and connecting them with properties. Possible errors are detected by a run of the validating service. For brevity's sake the usage of snapshots hereafter is described, not depicted as code.

A snapshot of a GeoTicTacToe instance in the pre game state will have a full set of nine actions with locations and token handlers according to the specification, as well as a set of eight TokenConditions forming the winning configurations. The resources are bound to an initial TokenDisperser separated into two sets, one for each player.

An in game snapshot will most likely show a progress in the game, as players at least have received their share of resources and carry it in their inventory. Therefore the initial dispenser is emptied and disabled. Depending on the players' progress, some caches have been captured as players have dropped resources and thereby occupied and disabled them for further interaction.

6 Discussion and future work

Goal of this work was to provide a representation for location-based games in a semantic ontology and facilitate semantic technologies to validate such snapshots. As the evaluation results in the previous chapter show (chapter 5), this has been mostly accomplished. The ontology and validation developed on top of W3C-WGS84, RCC8 and Pellet is adapted to represent location-based mobile games, although there is a lack of available solutions for qualitative spatial reasoning in ontologies as PelletSpatial has vanished. While the implementation of GeoTTT was straightforward and completely encodable in the ontology, some game state transitions of CityPoker and Neocartographer were at least partially not encodable.

The created ontology and validation toolkit is available under a free and open-source license.[1]

Nonetheless, there are many aspects that offer worthwhile improvements. First of all, there are some technical aspects which had to be left out in this work. There are issues regarding performance like the validation part augmenting the reasoner, where currently all restrictions in the knowledge base are queried and verified whereas a limitation on a given geogame would massively improve scalability. Another one would be the calculation of distances, currently being based on euclidean distances instead of a geographically based one. For modeling conditions like in CityPoker a valuation of resources would improve

[1] https://github.com/agp8x/geogame

the expressiveness of the ontology.

With a working CLI interface and JUnit tests for GeoTTT, a UI for other games is not that far. However, developing an interface suitable for any games presentable is another challenge. As the implementation of CityPoker and Neocartographer has shown, it might still be necessary to adapt some implementation code to represent some games, while others run well without further modifications.

The greatest drawback, however, is the limitation upon modeling snapshots of a game, leaving out temporal representation. One way to realize this could be obtained by using neighborhood relationships to relate snapshots according to their occurrence in the temporal game flow.

Bibliography

[1] B. Feulner and D. Kremer, "Using geogames to foster spatial thinking," 2014.

[2] R. Warren and E. Champion, "Linked open data driven game generation," in *The Semantic Web – ISWC 2014*, ser. Lecture Notes in Computer Science, P. Mika, T. Tudorache, A. Bernstein, C. Welty, C. Knoblock, D. Vrandečić, P. Groth, N. Noy, K. Janowicz, and C. Goble, Eds. Springer International Publishing, 2014, vol. 8797, pp. 358–373. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-11915-1_23

[3] I. Celino, "Geospatial dataset curation through a location-based game," *Semantic Web*, vol. 6, no. 2, pp. 121–130, 2015.

[4] C. Schlieder, P. Kiefer, and S. Matyas, "Geogames: A conceptual framework and tool for the design of location-based games from classic board games," in *Intelligent Technologies for Interactive Entertainment*. Springer, 2005, pp. 164–173.

[5] F. Echtler, S. Nestler, A. Dippon, and G. Klinker, "Supporting casual interactions between board games on public tabletop displays and mobile devices," *Personal and Ubiquitous Computing*, vol. 13, no. 8, pp. 609–617, 2009. [Online]. Available: http://dx.doi.org/10.1007/s00779-009-0246-3

[6] C. M. Olsson, S. Dahlskog, and S. Björk, "The conceptual relationship model: understanding patterns and mechanics in game design," in *DiGRA &#3914 - Proceedings of the 2014 DiGRA International Conference*. DiGRA, August 2014. [Online]. Available: http://www.digra.org/wp-content/uploads/digital-library/digra2014_submission_93.pdf

[7] S. Bjork and J. Holopainen, "Patterns in game design (game development series)," 2004.

[8] O. Davidsson, J. Peitz, and S. Björk, "Game design patterns for mobile games," *Project report to Nokia Research Center, Finland*, 2004.

[9] C. Will, "A pattern language for designing location-based games," Diploma Thesis, RWTH Aachen University, Aachen, June 2013.

[10] K. Janowicz, S. Scheider, T. Pehle, and G. Hart, "Geospatial semantics and linked spatiotemporal data–past, present, and future," *Semantic Web*, vol. 3, no. 4, pp. 321–332, 2012.

[11] “W3C Semantic Web Interest Group: Basic Geo (WGS84 lat/long) Vocabluary,” https://www.w3.org/2003/01/geo/, last accessed : 04.02.2016.

[12] G. A. Atemezing and R. Troncy, “Comparing vocabularies for representing geographical features and their geometry,” in *Terra Cognita 2012 Workshop*, 2012, p. 3.

[13] “W3C Geospatial Vocabulary,” https://www.w3.org/2005/Incubator/geo/XGR-geo-20071023/, last accessed : 04.02.2016.

[14] “NeoGeo Geometry Ontology,” http://geovocab.org/geometry.html, last accessed : 04.02.2016.

[15] C. Stadler, J. Lehmann, K. Höffner, and S. Auer, “Linkedgeodata: A core for a web of spatial open data,” *Semantic Web*, vol. 3, no. 4, pp. 333–354, 2012.

[16] S. Auer, J. Lehmann, and S. Hellmann, *Linkedgeodata: Adding a spatial dimension to the web of data*. Springer, 2009.

[17] L. Belouaer, M. Bouzid, and A.-I. Mouaddib, “Spatial knowledge in planning language,” in *International Conference on Knowledge Engineering and Ontology Development*, France, 2011, pp. –. [Online]. Available: https://halv3-preprod.archives-ouvertes.fr/hal-00960912

[18] D. A. Randell, Z. Cui, and A. G. Cohn, “A spatial logic based on regions and connection.” *KR*, vol. 92, pp. 165–176, 1992.

[19] I. Lee, “Fast qualitative reasoning about categories in conceptual spaces.” in *HIS*. Citeseer, 2003, pp. 341–350.

[20] Y. Katz and B. C. Grau, “Representing qualitative spatial information in owl-dl,” in *In Proceedings of the OWL: Experiences and Directions Workshop*, 2005.

[21] R. Battle and D. Kolas, “Geosparql: enabling a geospatial semantic web,” *Semantic Web Journal*, vol. 3, no. 4, pp. 355–370, 2011.

[22] M. Perry and J. Herring, “Ogc geosparql-a geographic query language for rdf data,” *OGC Implementation Standard. Sept*, 2012.

[23] M. Stocker and E. Sirin, “Pelletspatial: A hybrid rcc-8 and rdf/owl reasoning and query engine.” in *OWLED*, vol. 529, 2009.

[24] “OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax,” https://www.w3.org/TR/owl2-syntax/#Class_Expressions, last accessed : 04.02.2016.

[25] A. Garcia-Rojas, S. Athanasiou, J. Lehmann, and D. Hladky, “Geoknow: leveraging geospatial data in the web of data,” *Open Data on the Web (ODW13)*, 2013.

[26] T. Dohmen, M. Vreijling, T. Gornostay, R. Skadins, S. S. Tekiroglu, T. Can, and B. L. Linden, *The use of OWL for Spatial Knowledge based Applications*, 2010. [Online]. Available: http://www.semantic-web-journal.net/sites/default/files/SWJ73.pdf

A Appendix

A.1. Ontology models

```
Ontology(<http://clemensklug.de/uni/ba/geogame/geoTTT>
Declaration(Class(:GTTTstartCondition))
Declaration(Class(:GeoTTT))
Declaration(Class(:GeoTTTAction))
Declaration(Class(:GeoTTTDisperse))
Declaration(Class(:GeoTTTDraw))
Declaration(Class(:GeoTTTGroupLines))
Declaration(Class(:GeoTTTstartplace))
Declaration(Class(:GeoTTTtokenCond))
Declaration(Class(:GeoTTTwin))
SubClassOf(:GTTTstartCondition geogame:StartCondition)
SubClassOf(:GTTTstartCondition ObjectExactCardinality(1 geogame:has-condition geogame:PlayerLocationCondition))
SubClassOf(:GTTTstartCondition ObjectExactCardinality(1 geogame:has-condition geogame:TimeOutCondition))
SubClassOf(:GTTTstartCondition ObjectExactCardinality(1 geogame:satisfied-action :GeoTTTDisperse))
SubClassOf(:GeoTTT geogame:Geogame)
SubClassOf(:GeoTTT ObjectExactCardinality(9 geogame:game_action geogame:TokenAction))
SubClassOf(:GeoTTT ObjectExactCardinality(1 geogame:game_condition_draw :GeoTTTDraw))
SubClassOf(:GeoTTT ObjectExactCardinality(1 geogame:game_condition_start :GTTTstartCondition))
SubClassOf(:GeoTTT ObjectExactCardinality(1 geogame:game_condition_win :GeoTTTwin))
SubClassOf(:GeoTTTAction geogame:TokenAction)
SubClassOf(:GeoTTTAction ObjectExactCardinality(1 geogame:tokenhandler-is geogame:TokenCapture))
SubClassOf(:GeoTTTAction DataExactCardinality(1 geogame:question xsd:string))
SubClassOf(:GeoTTTDisperse geogame:TokenAction)
SubClassOf(:GeoTTTDraw geogame:DrawCondition)
SubClassOf(:GeoTTTDraw ObjectExactCardinality(1 geogame:has-condition geogame:TimeOutCondition))
```

```
SubClassOf(:GeoTTTGroupLines geogame:LogicCondition)
SubClassOf(:GeoTTTGroupLines ObjectExactCardinality(8 geogame:has-condition :GeoTTTtokenCond))
SubClassOf(:GeoTTTGroupLines DataExactCardinality(1 geogame:combination xsd:string))
SubClassOf(:GeoTTTstartplace geogame:PlayerLocationCondition)
SubClassOf(:GeoTTTstartplace ObjectExactCardinality(1 geogame:satisfied-action :GeoTTTDisperse))
SubClassOf(:GeoTTTtokenCond geogame:TokenCondition)
SubClassOf(:GeoTTTtokenCond ObjectExactCardinality(3 geogame:token-at geogame:TokenCapture))
SubClassOf(:GeoTTTwin geogame:WinCondition)
SubClassOf(:GeoTTTwin ObjectExactCardinality(1 geogame:has-condition :GeoTTTGroupLines))
)
```

Code Listing A.1: GeoTTT ontology without individuals

```
Ontology(<http://clemensklug.de/uni/ba/geogame/citypoker>
Declaration(Class(:CPStartCondition))
Declaration(Class(:CPfieldTokenSet))
Declaration(Class(:CPinit))
Declaration(Class(:CPinitHandler))
Declaration(Class(:CPinitSet))
Declaration(Class(:CPtokenDisperser))
Declaration(Class(:CPwin))
Declaration(Class(:CityPoker))
Declaration(Class(:CityPokerFieldAction))
SubClassOf(:CPStartCondition geogame:StartCondition)
SubClassOf(:CPStartCondition ObjectExactCardinality(1 geogame:has-condition geogame:LogicCondition))
SubClassOf(:CPStartCondition ObjectExactCardinality(1 geogame:has-condition geogame:TimeOutCondition))
SubClassOf(:CPfieldTokenSet geogame:TokenSet)
SubClassOf(:CPfieldTokenSet ObjectExactCardinality(2 geogame:has-token geogame:Token))
SubClassOf(:CPinit geogame:TokenAction)
SubClassOf(:CPinitHandler geogame:TokenDisperser)
SubClassOf(:CPinitHandler ObjectExactCardinality(2 geogame:dispersed-tokens :CPinitSet))
SubClassOf(:CPinitSet geogame:TokenSet)
SubClassOf(:CPinitSet ObjectExactCardinality(5 geogame:has-token geogame:Token))
SubClassOf(:CPtokenDisperser geogame:TokenDisperser)
SubClassOf(:CPtokenDisperser ObjectExactCardinality(1 geogame:dispersed-tokens :CPfieldTokenSet))
SubClassOf(:CPtokenDisperser DataExactCardinality(1 geogame:tokencount xsd:nonNegativeInteger))
SubClassOf(:CPwin geogame:WinCondition)
SubClassOf(:CPwin ObjectExactCardinality(1 geogame:has-condition geogame:TimeOutCondition))
SubClassOf(:CityPoker geogame:Geogame)
SubClassOf(:CityPoker ObjectExactCardinality(5 geogame:game_action :CityPokerFieldAction))
SubClassOf(:CityPoker ObjectExactCardinality(1 geogame:game_condition_start :CPStartCondition))
SubClassOf(:CityPoker ObjectExactCardinality(1 geogame:game_condition_win :CPwin))
SubClassOf(:CityPokerFieldAction geogame:TokenAction)
```

```
SubClassOf(:CityPokerFieldAction ObjectExactCardinality(3 geogame:at <http://www.w3.org/2003/01/geo/wgs84_pos#Point>))
SubClassOf(:CityPokerFieldAction ObjectExactCardinality(1 geogame:tokenhandler-is :CPtokenDisperser))
SubClassOf(:CityPokerFieldAction DataExactCardinality(1 geogame:question xsd:string))
)
```

Code Listing A.2: CityPoker ontology without individuals

```
Ontology(<http://clemensklug.de/uni/ba/geogame/neocartographer>
Declaration(Class(:NCDrawCondition))
Declaration(Class(:NCInitialDisperserAction))
Declaration(Class(:NCStartCondition))
Declaration(Class(:NCTokenAction))
Declaration(Class(:NCWinCondition))
Declaration(Class(:Neocartographer))
SubClassOf(:NCDrawCondition geogame:DrawCondition)
SubClassOf(:NCInitialDisperserAction geogame:TokenAction)
SubClassOf(:NCInitialDisperserAction ObjectExactCardinality(1 geogame:tokenhandler-is> geogame:TokenDisperser))
SubClassOf(:NCStartCondition geogame:StartCondition)
SubClassOf(:NCStartCondition ObjectExactCardinality(1 geogame:has-condition geogame:PlayerLocationCondition))
SubClassOf(:NCStartCondition ObjectExactCardinality(1 geogame:has-condition geogame:TimeOutCondition))
SubClassOf(:NCStartCondition ObjectExactCardinality(1 geogame:satisfied-action :NCInitialDisperserAction))
SubClassOf(:NCTokenAction geogame:TokenAction)
SubClassOf(:NCTokenAction ObjectExactCardinality(1 geogame:tokenhandler-is geogame:TokenCapture))
SubClassOf(:NCTokenAction DataExactCardinality(1 geogame:question xsd:string))
SubClassOf(:NCWinCondition geogame:WinCondition)
SubClassOf(:Neocartographer geogame:Geogame)
SubClassOf(:Neocartographer ObjectExactCardinality(1 geogame:game_condition_draw :NCDrawCondition))
SubClassOf(:Neocartographer ObjectExactCardinality(1 geogame:game_condition_start :NCStartCondition))
SubClassOf(:Neocartographer ObjectExactCardinality(1 geogame:game_condition_win :NCWinCondition))
SubClassOf(:Neocartographer ObjectMaxCardinality(16 geogame:game_action :NCTokenAction))
)
```

Code Listing A.3: Neocartographer ontology without individuals

List of Figures

2.1. Overview of games design patterns relevant for location-based games [9] . . 6
2.2. Abstraction levels of game design [6] 7
2.3. Conceptual relations [6] 7
2.4. SpaceOntology [17] 9
2.5. RCC8 [18] 10

4.1. Structure of the ontology 19
4.2. Caches and actions 20
4.3. Resources and their usage 22
4.4. Dynamic elements 23
4.5. Model for players 23
4.6. Main individual representing a concrete game instance 24
4.7. Class diagram Validation 26

List of Tables

3.1. Features of spatial representations 14
3.2. Evaluation of game design patterns regarding relevance for a geogame model 16

5.1. Ontology metrics 30

List of Algorithms

4.1. Validation algorithm 25
4.2. Spatial validation algorithm 25
4.3. Average group distance calculation 26

List of Abbreviations

API	Application Programming Interface
CLI	Command Line Interface
GeoTTT	GeoTicTacToe
GPX	GPS Exchange Format
JSON	JavaScript Object Notation
OGC	Open Geospatial Consortium
OWL	Web Ontology Language
OWL 2 DL	Web Ontology Language 2 Direct Semantics
OWL DL	Web Ontology Language Description Logic
RCC	Region Connection Calculus
RDF	Resource Description Framework
SPARQL	SPARQL Protocol and RDF Query Language
W3C	World Wide Web Consortium
XML	Extensible Markup Language

List of Symbols

$A : \{a \mid a : S \to S\}$	Actions [4]
$G : (S, A, time, value, sync)$	Geogame [4]
L	Location [4]
P	Player [4]
R	Resource [4]
$S : \{s \mid s : (P \to L, R \to L \cup P)\}$	States [4]
$time : A \to \mathbb{R}^+$	Temporal coherence [4]
$value : S \to V$	Valuation [4]

Eidesstattliche Erklärung

Ich erkläre hiermit gemäß § 17 Abs. 2 APO, dass ich die vorstehende Bachelorarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Bamberg, February 24, 2016 ....

Clemens Klug